



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 17/00, 17/30, 19/00	A1	(11) International Publication Number: WO 95/11487
		(43) International Publication Date: 27 April 1995 (27.04.95)

(21) International Application Number: PCT/US94/12074

(22) International Filing Date: 24 October 1994 (24.10.94)

(30) Priority Data:
08/141,285 22 October 1993 (22.10.93) US(71) Applicant: FDC, INC. [US/US]; 600 South Highway 169,
Minneapolis, MN 55426-1209 (US).(72) Inventors: EMERSON, Michael, Gene; 16105 Baywood Lane,
Eden Prairie, MN 55346 (US). WESTMAN, Kelly, Reed;
5212 Abbott Avenue South, Minneapolis, MN 55410 (US).
PILLAI, Sushil; 4384 Hamilton Avenue, Eagan, MN 55123
(US).(74) Agent: BRUESS, Steven, C.; Merchant, Gould, Smith, Edell,
Welter & Schmidt, 3100 Norwest Center, 90 South Seventh
Street, Minneapolis, MN 55402 (US).

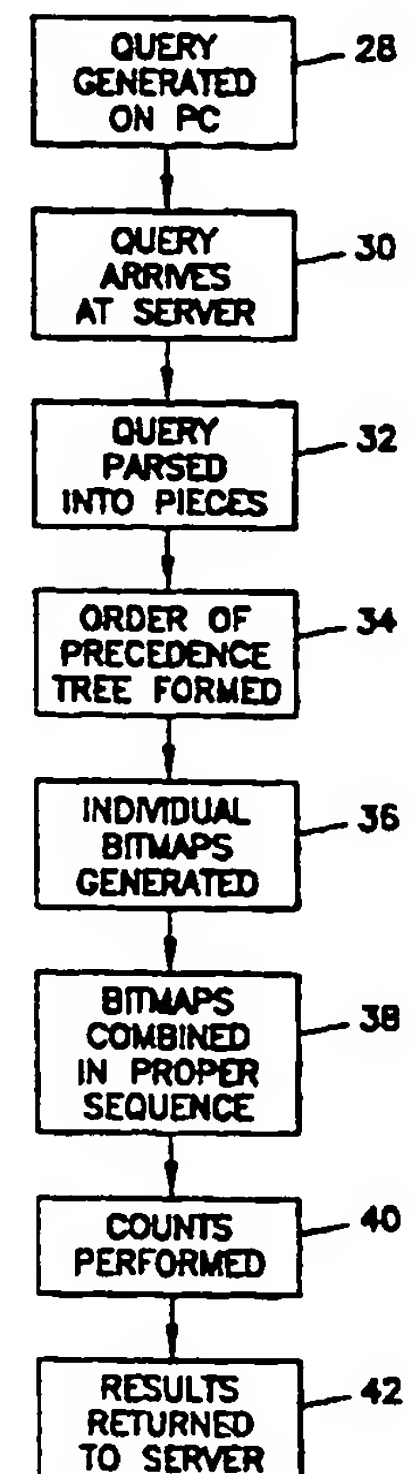
(81) Designated States: CA, GB.

Published*With international search report.**Before the expiration of the time limit for amending the
claims and to be republished in the event of the receipt of
amendments.*

(54) Title: DATABASE USING TABLE ROTATION AND BIMAPPED QUERIES

(57) Abstract

A system to enhance query performance. The query is performed on a modified database in which data entries for each field are stored contiguously across all records. The query is parsed (32) into portions directed to each field referenced by the query. Each portion is searched over the contiguous entries of the appropriate data field. A bitmap is generated (36) by producing a string of bits, each bit representing a record, and setting bits corresponding to records that satisfy the query. The resulting bitmaps are combined (38) to produce the results of the original query. The preferred embodiment is a direct marketing database on a client server system using a spreadsheet like interface for query manipulation. Data segmentation, ad hoc requests and systematic research are allowed. A suite of reports specific to the needs of direct marketers may be produced on a regular basis.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	GB	United Kingdom	MR	Mauritania
AU	Australia	GE	Georgia	MW	Malawi
BB	Barbados	GN	Guinea	NE	Niger
BE	Belgium	GR	Greece	NL	Netherlands
BF	Burkina Faso	HU	Hungary	NO	Norway
BG	Bulgaria	IE	Ireland	NZ	New Zealand
BJ	Benin	IT	Italy	PL	Poland
BR	Brazil	JP	Japan	PT	Portugal
BY	Belarus	KE	Kenya	RO	Romania
CA	Canada	KG	Kyrgyzstan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo	KR	Republic of Korea	SE	Sweden
CH	Switzerland	KZ	Kazakhstan	SI	Slovenia
CI	Côte d'Ivoire	LI	Liechtenstein	SK	Slovakia
CM	Cameroon	LK	Sri Lanka	SN	Senegal
CN	China	LU	Luxembourg	TD	Chad
CS	Czechoslovakia	LV	Latvia	TG	Togo
CZ	Czech Republic	MC	Monaco	TJ	Tajikistan
DE	Germany	MD	Republic of Moldova	TT	Trinidad and Tobago
DK	Denmark	MG	Madagascar	UA	Ukraine
ES	Spain	ML	Mali	US	United States of America
FI	Finland	MN	Mongolia	UZ	Uzbekistan
FR	France			VN	Viet Nam
GA	Gabon				

DATABASE USING TABLE ROTATION AND BITMAPPED QUERIES

BACKGROUND OF THE INVENTION

In the direct mailing business, companies
5 maintain very large databases of customers that might
range from 1 million to 20 million records. Because
these files are so large the direct marketing industry
has grown up with very large and simplistic database
systems rather than taking advantage of the newer
10 relational technology that is available. By relational
technology, it is meant the ability to interrelate
various types of information together in a dynamic
fashion so that various information about a customer,
for example demographics, what products they purchased,
15 when they last purchased, etc. may be interrelated.

Many other industries that use databases for
storage of large amounts of information concerning
customers have been able to take advantage of relational
technology. In some of those industries, for example
20 retail stores, the relational database has worked
effectively by segmenting the data into chunks, so that
all 2 million of the retail store's customers do not
have to be looked at each time a search is performed.
Customers can be grouped by individual stores. Another
25 example would be in the banking industry, where banks
segment their data such that only customers of one of
its branches are reviewed. On the other hand, large
direct marketing companies do not segment up their
customers. They want to look at patterns that cross all
30 of its millions of customers, making it very difficult
to segment up a direct marketing database into chunks
while maintaining effective access to the data. At
present, a large direct marketing company would
typically store its data in large files with one record
35 for each customer. The record would typically be stored
on a tape or a mainframe. To illustrate the structure
of data stored in the standard relational database, take
a database of 1 million customers that contains at least
the information on state, zip, age, income, city and

gender. A standard database stores data in representative fields within a record for each customer as shown in FIGURE 1. Each of the six categories shown are representative of one field. In this example if it is desired to get three points of information, gender, state, and income level, a search of all records for all customers must be performed.

In traditional technology the progression of this search is to pull the first record off the file, extract the three pieces of data from the record, load such data into a table, and proceed to the next record. This process would be repeated 1 million times in a case such as that shown in FIGURE 1 having 1 million customers/records. This is a very inefficient process and in the direct marketing context where there could be as many as 750 to 2,000 fields the search can become very time consuming for very simple projects such as the above example.

The computer accesses the queried information in traditional technology by grabbing the entire record, which in some instances may include as many as 750 to 2,000 fields. The record is stored in computer memory and the desired fields, three in this example, are reviewed and followed by a determination of whether this particular record satisfies the query. If the above example had involved 2000 fields and 10 million customers, that would be 20 gigabytes of information that the computer would have to pass through to answer the query. This is a very inefficient process, when considering that most direct marketing data bases are of this size and that most queries involve an average of 6 to 10 fields out of the thousand fields being searched. Therefore, in standard data base searching, when a record is brought into the computer and data is downloaded, there is a large inefficiency ratio when comparing the amount of data searched to that which is actually used.

There are two issues that the present invention attempts to address. First, the architecture of the very large files in direct marketing make it very expensive to answer even the most simple question. So whether the query would acquire a lot of information or is as simple as how many females bought a particular product, the standard database requires that all the records be searched to find such information. This is a very slow and expensive process. The present invention provides a tool that shortens the search time for simpler tasks, performing some in a matter of seconds.

This invention, Database Link™, relates to a system designed to give rapid access to large relational marketing databases. In particular, it is a product designed specifically for professionals in direct marketing who desire to get mission-critical information from large marketing databases. These databases typically have anywhere from 1 million to 25 million customers plus up to 10 times that many additional detail records.

In addition to the large size of these databases, direct marketing has several unique characteristics that make getting information from these files particularly challenging. First, the databases are quite homogenous. As opposed to databases in industries such as banking and finance that can logically organize customers into geographic "lumps", direct marketers look at their customer base in a more monolithic fashion. This reduces the effectiveness of common database strategies which look to segment files to improve response time.

Second, these databases contain a large and growing amount of information on each customer. Direct Marketers are moving towards finer and finer targeting of their promotions which requires huge amounts of information about individuals and households. It is not uncommon to store upwards of a thousand fields on each

individual, and the amount of fields will continue to increase at a rate of 25 to 50 per year. This fact, combined with the first point about the monolithic nature of these databases, creates a huge challenge to today's hardware and software technology. Within this challenging environment, Database Link™ is designed to meet the needs of direct marketing professionals.

SUMMARY OF THE INVENTION

10 According to the invention, a method and system are provided that allow direct marketing personnel to reduce the time and enhance the efficiency of searches performed on direct marketing data records. The user enters a query on an IBM compatible PC running
15 a client server program under the Microsoft Windows™ environment. After the query has been entered, the client server turns the query into a packet which is sent to the system server.

The server stores data from a standard
20 database in modified form, rotated 90 degrees. Instead of data being stored contiguously for each individual customer listed in the database, as is done in a standard database, the data for each field across all records in the database is stored contiguously.

25 The queried information is retrieved and processed using a process called bitmapping, which reduces the search time per the complexity of the query in contrast to the standard database system. After data has been captured it is packetized and returned to the
30 client server program where it can be reviewed.

The present invention can also produce a suite of reports that are specific to the needs of direct marketers. These include reports on RFM information (recency of last purchase, frequency of purchases, and
35 monetary totals of life to date information. These reports are generated on a regular (usually monthly

basis) and are used to drive the direct marketing process.

The present invention also provides a system that permits data segmentation, **ad hoc requests**, and systematic research.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is an illustration of data records stored in a relational database in contiguous form for each individual customer listed in the database;

10 Figure 2 is a simplified schematic drawing of the Database Link query system;

Figure 3 is an illustration of data records stored in a relational database after 90 degree rotation and reprocessing so that data stored is contiguous for each field across all records;

Figure 4 is an illustration of the file structure types used by the present invention;

Figure 5 is an illustration of the spreadsheet of the client server windows software according to the present invention;

Figure 6 is an illustration of the Build Query box screen of the client server windows software according to the present invention;

Figure 7 is an illustration of the Distribution Screen of the client server windows software according to the present invention, which allows for the selection of any of the tables contained in the database and any set of ranges;

Figure 8 is an illustration of the data tabulation screen of the client server windows software according to the present invention;

Figure 9 is an illustration of the data tabulation screen of the client server windows software according to the present invention;

35 Figure 10 is an overview block diagram of the query processing of the present invention; and

Figure 11 is an overview block diagram of one method of combining multiple bitmaps resulting from a complex query.

5

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

In the following detailed description of the preferred embodiment, reference is made to the accompanying drawings which form a part hereof and in which is shown by way of illustration an exemplary embodiment in which the invention may be practiced. This embodiment is described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other
10
15
20
embodiments may be utilized and that structural or logical changes may be made without departing from the scope of the present invention. The following detailed description is, therefor, not to be taken in a limiting sense, and the scope of the present invention is defined by the appended claims.

For purposes of this application, the stored data that is referred to throughout, is relational database data that has been modified. The relational database data is modified by rotating the data files 90
25
degrees. Now, instead of data being stored contiguously for each individual customer listed in the database, the data for each field across all records in the database is stored contiguously. In short, each individual record instead of being in columnar form is now stored
30
in rows and each field previously stored as rows is now stored in columnar form.

The Database Link™ system, which is the product name for the preferred embodiment of the present invention, is based on a client server system design.
35
In this paradigm, the processing for the user interface is on a separate computer from the system that actually accesses the data and produces the result that has been requested.

Referring now to FIGURE 2, an overview of the invention will be described. The user interface piece resides on an IBM compatible PC 10 and runs under the control of a client windows program in the Microsoft
5 Windows™ environment. The client windows program is built around a sophisticated grammar that models the real world environment of the database marketer. It assists the user in creating queries that are turned into packets and sent to the server 14 processor over a
10 LAN or telephone wire based WAN 12.

Once the server 14 has received a request, it begins to process the data until the results have been completed. The server 14 uses an approach built on
15 bitmaps providing users with the ability to rapidly retrieve data. After the requested data is retrieved, the server 14 puts the results into a packet handling system that delivers the results back to the PC 10 that requested the information.

20 The Client Windows Program

The client windows program exists to allow easy interface between the user and the database server.

It is based on a basic spreadsheet type of
25 paradigm within the Microsoft Windows environment. The application is designed to be compatible with Windows and meets all of the requirements as specified by Microsoft in their Windows Applications Development document.

30 The client windows program allows for easy formulation of queries and "packetizes" the information and sends it to the database server for processing. There are two main components to the client windows program: a language parser that enables parsing of
35 complex marketing queries and a spreadsheet/report specification that allows many queries to be organized into one query.

The language parser interprets the DBL grammar. The DBL grammar is based on an industry standard SQL database query specification and adds specific functionality to meet the needs of the Direct Marketing industry.

The Spreadsheet

The spreadsheet is what the user sees when entering the Database Link™ application, illustrated in FIGURE 5. The spreadsheet allows for the optional storage of queries into batches for submission in one "lump" to the server 14. Each row in the spreadsheet represents one group of customers that are to be selected.

The first two columns are for labeling purposes. The keycode column is a special mnemonic used by marketers for naming groups and the labels column is any label that the user wants to identify a group. Each additional column contains one criteria that defines that particular group. Each of these criteria is "ANDED" together to define a group. Thus, if the first column contained "gender is male" and the second column contained "state is Minnesota", the result for that row would be all males who are from the state of Minnesota.

In addition to this basic functionality, the spreadsheet has all of the standard spreadsheet functionality built into Windows applications. This includes the capability to cut, copy, and paste cells, rows, and columns. Rows and columns may be inserted as desired. Database Link™ allows the user to proof the queries contained within each cell to make sure they are grammatically correct queries.

Building Queries

The user is provided a "point and click" dialog box for the actual creation of queries to send to the database. The Build Query box is shown in FIGURE 6.

These queries can be sent directly to the database for processing or they might be saved to the spreadsheet where they might be submitted in a batch.

The Build Query dialog box moves the user through several steps starting with the selection of Query Table in the far upper left-hand part of the box. Once the table has been selected, all of the fields that are contained in that table appear in the second line which is currently empty. Once the field has been selected, a list of valid operators appears in the third line. Finally the user enters the actual value in the fourth line. Pieces of queries are then accumulated into a total query in the bottom spreadsheet connected with and's/or's as selected on the far left side of the box.

Distributions

Database Link™ allows the user to find out additional information about customers beyond straight-forward counts. For example, if the above user wanted to find out the most recent purchase dates of the male Minnesotans, the distribution option would allow the user to select those customers and get a distribution of results on any of the fields in the database. The Distribution Screen is shown in Figure 7. This screen allows for the selection of any of the tables contained in the database and any set of ranges (e.g., a date field may be lumped into months or weeks).

30 Two and Three Way Crosstabs

In addition to distributions on a single field, Database Link™ also allows for results to be tabulated across two and three fields. The setup for this is shown in Figures 8 and 9. A table is chosen which defines a set of fields to choose from. This is followed by a specification of ranges for the field chosen.

Client ServerHardware Specifics

Database Link™ server is currently optimized to run on a Digital Equipment Corporation (DEC) Alpha AXP server under the VMS operating system. The program is written in C and the core processes are nearly independent of hardware or operating system platform. The exceptions to the above are two-fold:

1. **DecMessageQue** (DMQ) is a messaging system that reliably moves information from one computer to another. Database Link™ uses this product to get information from the PC 10 to the server 14 and results from the server 14 back to the PC 10. This product is integral to the overall functioning of Database Link™. However, it has no impact on the uniqueness of patentability of the product. Other messaging systems are available that could provide similar or identical functionality. The system could easily be migrated to another product or to an entirely different hardware/software platform with no changes in functionality.
2. **Global Sections.** DEC VMS has a unique feature called global sections that allows the software engineer to map large sections of a disk file directly to an area in computer memory. Once this is done, any access to this area causes automatic swapping of data from disk to memory on an as needed basis. Essentially, this allows a program to contain multiple gigabytes of data that appear to be in memory at one time even though physical memory is only a fraction of this amount. A useful analogy would be to think of catching a hundred pound fish with a five pound test line.

Database Link™ uses this capability throughout its core functions. Rather than performing inefficient reads from the disk, the area of data that is needed is "mapped" to a global section and

the operating system optimizes the access to this information. Because this operating system is very efficient at this type of memory optimization and swapping, this approach is very fast. It also
5 allows an inherent multi-threading (more than one type of supporting process happening at the same time) that enables disk I/O to happen at the same time as data is evaluated at the beginning of a process.

10 Data Base Link evaluates data in a manner different than that which occurs in a standard relational data base. The data evaluation process is different because data is not stored in the server 14 of the present invention as it is normally stored in
15 standard relational databases.

Data Structure

In standard relational databases, data is gathered and stored contiguously. In most cases, this contiguously stored data is in the form of a columnar
20 record as illustrated in FIGURE 1. The record represents the data stored on each individual direct marketing customer. It stretches across numerous fields, such as state, zip, age, income, city and gender.

25 When data within a standard relational database is modified for storage on the Database Link™ server, essentially, each record is removed, rotated by 90 degrees, and placed within the memory of the Database Link™ server 14. Rather than having all data stored
30 contiguously for each individual customer, as shown in FIGURE 1, the data for each field across all records/customers in the database is stored contiguously as illustrated in FIGURE 3. For example, instead of having columns that are one thousand fields long for
35 each of 1 million customers, as shown in FIGURE 1, the structure has been modified to have one thousand columns (fields) that are 1 million bytes long, as shown in

FIGURE 3. Each column, as shown in FIGURE 3, represents one field of data across all customers.

Effectively, this restructuring of data allows for simple questions, such as, for example, how many
5 customers who are female, live in the state of Minnesota, and make over \$25,000, to be answered by simply going through three columns (fields) of information, rather than all of the records for all of the customers which is done in a standard database query
10 environment.

To illustrate this fundamental difference, we look to FIGURE 1, illustrating a standard database of 1 million customers. Performance of a query, in accordance with the above example, of how many customers
15 are female, live in the state of Minnesota, and make over \$25,000, illustrates the difference between standard database data structure and that of Database Link™. Because the data is stored contiguously for each individual, as a record in a standard database, the
20 information desired is only several (or at the most several hundred) bytes apart on the hard disk. The standard database system will read in the entire record for each customer, decode the 3 fields in question and make an evaluation as to whether or not the record in
25 question meets the criteria that has been defined. This operation would be repeated for the million times that is necessary to complete the file. Specifically, the data for State followed by Zip, Age, Income, City, Gender and the remaining fields will be reviewed for
30 each of the 1 million customers to determine which customers satisfy the query. If there are 1000 fields of data, there will be 1 billion pieces of data (1000 fields * 1 million customers) reviewed.

On the other hand, in the Database Link™
35 structure, illustrated in FIGURE 3, these three pieces of information are likely to be millions of bytes apart in the data file for any one individual. Searching back

and forth to each field for each individual would totally negate the advantages that are inherent in the columnar data approach. The approach that Database Link™ makes is to evaluate all of the decisions for one
5 field at a time. These results are then inter combined using a technique called bitmaps.

Specifically, when the above example query is performed in Database Link™, all of the data for the 1 million entries for state will be reviewed, followed by
10 a review of the 1 million entries for income, followed by a review of the 1 million entries for gender. The total amount of data reviewed in the Database Link™ system is 3 million pieces of data as compared to the 1 billion pieces of data as shown for the standard
15 database system.

This difference becomes meaningful in marketing databases, when there are millions of records and thousands of fields, as in the example above, illustrated in Figures 1 and 3. This example shows that
20 the Database Link™ system reduced the amount of data reviewed by 997 million pieces, when compared to a standard database system. This reduction of data reviewed translates into faster query result times.

Because Direct Marketing databases involve
25 large files, even simple queries, such as the above example, result in quite inefficient operations in the standard database system. In an attempt to reduce the inefficiencies, standard database technology uses keys (indices) to get faster access to a particular record in
30 the file. These keys are stored and separated from the data and allow a database system to make certain decisions without reading the entire record. If the query that a user wants to make to a file involves data that is keyed, then the data can be accessed quite
35 quickly.

Keys work very well in databases where nearly all of the queries are done against a few significant

fields. For example, a customer service database can have keys by customer name, account number, and an order number. Information can then be retrieved by these keys in an almost instantaneous fashion. The problem with
5 keys is that they require significant amounts of disk overhead to maintain. Keys can require up to 10 bytes of space for each record for each key. Thus, ten key fields on a 1 million customer file costs an additional 100 megabyte of disk storage.

10 Columnar (Inverted) File Structure

Database Link™ solves this problem by doing away with keys altogether. Rather than speeding up file access by setting up specific keys, Database Link™ speeds up file access by segmenting all of the
15 information from one field together into one spot on the disk, thus dramatically reducing I/O and simplifying the process of evaluating criteria for reports.

This columnar file structure dramatically reduces the amount of I/O that is required for a
20 particular query or set of queries. With this type of data structure, the Database Link™ server can scan a given query and identify which fields are necessary. A look-up table defines where this information is located and then only this particular part of the database has
25 to be scanned to answer the particular query.

The challenge of this type of approach is in how to combine results across multiple fields, once they have been designated. The present invention uses a technique called bitmapping to address this problem.

30 Bitmaps

A bitmap is a series of computer words strung together in a one dimensional array. It looks at data as a series of bits rather than a higher level data type such as an integer or floating point value. A bitmap
35 is viewed as a uniform series of bits and within that bitmap, the word boundaries that are normally meaningful in how the computer "chunks" up its information are

meaningless. Each bit represents a piece of data that can be a yes or a no.

Bitmaps are used to mark all records within a particular column that meet a criteria. For example, if
5 the criteria is "gender is male", then a bitmap is created that has as many bits as there are records for that particular database table. For each record where the gender attribute is set to male, the corresponding bit in the appropriate bitmap is set to a 1 = yes.

10 The major advantage of bit level data is that it allows for the storage of huge amounts of yes/no type of information within a relatively small amount of space. Because of the eight to one ratio of bits to bytes, a database with 2 million customers can be
15 represented with 250,000 bytes of internal data.

Bitmaps provide another advantage to the Database Link™ paradigm: they can be combined together in a very rapid fashion to provide complex results. Very few database queries actually consist of only one
20 criteria as in the above "gender is male" example. Most queries consist of many criteria that are combined in some fashion using boolean arithmetic to arrive at a final subset of customers that is meaningful to the requester of information. In these types of cases,
25 several bitmaps must be combined together to form some type of aggregated result. Figure 11 illustrates one method of combining multiple bitmaps resulting from a complex query.

For example, for the query select "all males
30 who live in the state of New York", two bitmaps can be computed--one for all males and one for the residents of the state of New York. The answer to the composite of these two criteria can be derived by doing a very fast bit-level AND operation. With current architecture
35 supporting 64 bit operations, data is stored in chunks of 64 bits. This allows for one CPU instruction to perform the equivalent of 64 AND operations if they were

performed on a record by record level. Thus, for the query to select all males who live in the state of New York, rather than combining the two bitmaps generated from the 2 fields, the computer actually combines the 2
5 bitmaps in chunks of 64. Logically, however, bitmaps are combined bit by bit. For instance, customer X meets the query criteria if the bit in each bitmap corresponding to the customer X is set. However, the computer as explained above combines the bitmaps 64 bits
10 at a time.

The 64 bit processing performed by the computer speeds up the process of determining whether a particular customer satisfies the query while using less memory. In contrast, standard relational database
15 processing examines an entire record to determine if the query is satisfied, and then proceeds to the next record. This standard method required one operation for each record. Since the present invention processes 64 bits at a time rather than one operation for each
20 person, processing can be performed for 64 people in parallel. This produces a dramatic reduction in the amount of work that the computer has to do. This is possible due to the fact that the server looks at words multiple bits at a time. Therefor because 64 bits are
25 stored as a word, when bitmaps are combined it can be determined whether 64 people satisfy a given criteria for one combination.

It is only after the computer has done word combinations that a searcher can go back and look
30 through at each bit to determine whether a customer actually met the criteria and very quickly get a count of how many actually met the criteria.

The one challenge that must be met when using this approach is how to count bits within a bitmap in a
35 very efficient fashion. All of the above gains would be lost if it required large amounts of resources to count the bits that are contained in a results bitmap. The

present invention includes a procedure whereby a mapping algorithm is used to take 16 bits at a time, convert them to an integer value (0 to 65,611) for use as an index into an array of that length that stores the number of bits contained within that 16 bit segment. This approach allows for very rapid bit counting across very large bitmap arrays.

Iterative Capability

Many of the procedures that marketing professionals perform are iterative in nature. One query provides information that is used to further define a particular query. For example, a company may have a budget to mail 100,000 catalogs to its customers and the marketing director needs to define the optimal target audience to optimize the value of this mailing. The user of the system would like to test different date and dollar range combinations to produce this exact number. Database Link™ models this real world situation by storing a "ring" of the 20 (or whatever number optimizes performance with the resources available) most recent queries that have been used by a particular user. Each node on the ring contains query definition as well as a bitmap of those records that met that particular criteria. As new queries are entered, they are broken down into pieces and each piece is matched against the queries that are stored in the ring. If a match is found, then the query does not have to be reexecuted. The present invention uses a basic FILO (first in, last out) algorithm for tracking which queries to store in memory. To further optimize the performance of the overall system, more sophisticated weighting algorithms similar to those used in contemporary operating system memory swapping algorithms could be used.

FIGURE 11 is illustrative of a complex query having multiple bitmaps 41-48. Each bitmap 41-48 stores the result of a particular query on a field of data. These bitmaps can be combined by the CPU using logical

operations such as AND and OR to produce increasingly complex query results 49-55. As explained above, the processing of this bit information is extremely efficient since a single CPU instruction can operate on the bits corresponding to 64 customers in parallel. If bitmaps 41-48 are stored on the ring, and a new query matches bitmap 43, then this new query is not recalculated. The elimination of this processing further reduces search time.

10 Relational Structure

Database Link™ has made a third advance in making this technology work for marketing applications: it allows for relational marketing queries. As shown in FIGURE 4, most of the applications that use any of the unique types of data storage contained in Database Link™ are flat-file paradigms: they enable fast and efficient queries against one rectangular file, not several files joined together in a more real world representation of data. Database Link™ actually stores inverted indices to gain access across multiple data tables.

Complex Bitmap Processing

Customer Level

As shown in FIGURE 4 Database Link™ involves a number of different file structure types. In the embodiment illustrated in FIGURE 4 there are a number of different file types: customer level 20; subsidiary level 22-24; purchase table 26; product/line item detail 28; and promotion history 30.

30 Customer level information 20 is the hub of the Database Link relational structure. The data is stored with one record of information for each customer. The information stored within the customer summary file as shown in FIGURE 4, would be information uniform to all listed customers. For example all customers have an age, income level, gender and state.

Subsidiaries

Subsidiaries 22-24 represent data that is present for a portion of the customer table, thus it represents a one to few relationship. Each subsidiary
5 is mapped back to the customer record with a bitmap that contains one bit set for each record in the subsidiary table that matches to the customer record.

Purchase table

10 The purchase table 26 contains multiple records for each customer. Each record represents one activity transaction made by a particular customer. The purchase table is linked to the customer level table by an index that contains the number of purchases for each
15 customer. Customer records and purchase records can be joined together by starting at the top of the respective columns and referencing the appropriate element within the purchase column.

20 Product/Line Item Detail

Product level detail 28 is stored in a similar fashion to purchase level data. The additional complexity is that this level of information is many-to-one with customer as well as with purchase level
25 information. Two index arrays are used: one for customer to product level and the other for purchase to product level.

Promotion History

30 Promotion history information 30 is stored in the same format as purchase level information. Each record represents one promotion event.

Data Types

Database Link supports a variety of data types. Where possible, all data is stored in a binary format to minimize storage requirement and maximize throughput.

Characters

Character data is stored in a single byte, ASCII character set format. Data may only be in the range of valid ASCII printable character set. At the present time, Database Link stores all alphabetic data in an upper-case format.

The operators that are available for all character fields are shown in Table 1.

TABLE 1: Character Fields

is	is equal to any member of the list of values that follows
is not	is not equal to any of the members of the list of values that follows

Strings

Strings are fields that contain 2 or more characters for a particular field.

Because of the wide use of strings that have a certain finite number of valid formats (e.g., catalog numbers), Database Link uses a hashing function/look-up approach to storing this information as an integer pointer to a table that contains the actual values. This approach allows the storage of up to 65000 unique values in a two-byte integer field. Given that typical strings of this type are 8 to 12 bytes in length, this technique provides an average of 5:1 data compression without any loss of data. Additionally, this data approach allows for integer word comparisons instead of byte by byte string comparisons that require many additional CPU cycles.

Valid operators for strings are as shown in Table 2.

5

TABLE 2: Valid Operators

is	is equal to any member of the list of values that follows
is not	is not equal to any of the members of the list of values that follows

10

15

Integers

All integers are stored in a standard binary format of 1, 2, or 4 bytes in length. This allows for maximum efficiency in terms of access and storage.

20

Valid operators for integers are as shown in Table 3.

25

TABLE 3: Valid Operators for Integers

>, <, >=, <=, =	Basic boolean numeric operations
over	greater than or equal to the value following
under	less than the value following
is	is equal to any member of the list of values that follows
is not	is not equal to any of the members of the list of values that follows

30

35

Dollars

Dollars are stored as integers with number of cents. Operators are the same as for integers.

Dates

40

All dates are stored in number of days since January 1, 1888. This modified Julian type of approach allows for dates between 1888 and 2064 in a two-bytes

(16 bit) integer format. Storing dates in this fashion allows for efficient operations.

Valid operators for dates are as shown in Table 4.

TABLE 4: Valid Operators for Dates

	>, <, >=, <=, =	Basic boolean numeric operations
10	after	greater than or equal to the date following
	before	less than the date following
	within	DBLink has a special capability to select all records that are within a certain number of days, weeks, or months of today's date
	during	DBLink has the capability to select based on a seasonal basis: all records within a certain season across a certain number of years
	is	is equal to any member of the list of dates that follows
15	is not	is not equal to any of the members of the list of dates that follows

20 Floating Points

Floating point values are stored as integer to reduce space and increase processing efficiency. An implied decimal point is stored with the data to allow for various decimal precisions. Operators are the same as for integers.

Bitmaps Defined

In Database Link™, bitmaps are used to mark all records within a particular table that meet a criteria. For example, if the criteria is "gender is male", then a bitmap is created that has as many bits as there are records for that particular database table. For each record where the gender attribute is set to male, the corresponding bit in the appropriate bitmap is set to a 1.

Creating Bitmaps

Many marketing queries consist of a list of values rather than a specific single value. For example, the marketing question: *How many customers*
5 *made their first purchase during Christmas season over the past 3 years?* would translate into a series of ranges that might be as follows:

First_order_date is 10/15/91-12/31/91, 10/15/92-12/31/92, 10/15/93-12/31/93

10 A standard SQL database would reduce this to a series of OR queries with each individual range translated into a between construct. This type of approach would be very inefficient for Database Link which must go through an entire column for each of the
15 three sub-segments of the query. Thus, Database Link includes a facility for evaluating a complex set of ranges and unique values in one pass of the data file. A list can consist of a series of values separated by commas or dashes. The comma signifies a unique value
20 and a dash signifies all values between the value in front of the dash and behind it. Because all of these values are handled as one criteria, it is the equivalent of having a series of ORs representing each comma and the entire expression enclosed in parenthesis, thus
25 forcing it to be evaluated in one lump for each individual in the table.

Database Link also supports a variety of wild card conventions for all string types. The most straight-forward of these capabilities include the
30 following four wild card characters:

*, #, @, ?

The asterisk (*) provides a means of matching all characters to the end of a particular string. A pound sign (#) matches a single numeric (digit only)
35 character. The at sign (@) matches a single alphabetic character (A-Z). Finally

Field Comparisons as a Special Case

In all of the above examples, the criteria has been a fixed value. That is, a field of variable information is compared to a specific fixed value or
5 list of values. In some cases, it may be desirable to compare two fields together to find a particular result. For example, the question: *How many customers purchased more on their most recent purchase than they did on their first purchase?* can only be answered by comparing
10 the two pieces of data together in a pair-wise type of fashion.

Cross Table Comparisons

Database Link has the capability of comparing
15 information across tables in various types of join operations. Each of these comparisons is done by combining bitmaps of specific within table information together in a particular fashion.

20 Subset Comparisons

Subset comparisons involve the synchronization of data between two columns where the data in each column does not match one to one with the data in the comparison column. Database Link handles these
25 comparisons by using the customer level information as a reference. Database Link maintains a bitmap for each of the subsidiary tables as to which data records are contained in that particular table compared to the customer table.

30

Many-to-one Comparisons

Many-to-one comparisons are much more complex in Database Link than other types of comparisons. Many to one comparisons allow for the joining together of one
35 customer with one or more purchase/activity records of one or more promotion types of records.

Patterned Comparisons

The above descriptions apply to basic many-to-one comparisons. Database Link also has the capability to apply various patterns of comparisons to this comparison. Thus, rather than looking for the presence of any records that meet a particular criteria, the user may want to know whether the first record only meets a criteria. The types of comparisons that Database Link allows are shown in Table 5.

TABLE 5: Database Link Comparisons

15	First	The first record must meet the criteria specified. If no records exist, then the comparison fails.
	Second	The second record must meet the criteria specified. If a second record does not exist, then the comparison fails.
	Third	The third record must meet the criteria specified. If a third record does not exist, then the comparison fails.
	Last	The last record must meet the criteria specified. If no records exist, then the comparison fails.
20	Next to last	The next to last record must meet the criteria specified. If at least two records do not exist, then the comparison fails.
	After first	At least one record after the first record meets the criteria specified. If only one record exists, then the comparison fails.
	Last 2	At least one of the last two purchases must meet the specified criteria.
	Last 5	At least one of the last five purchases must meet the specified criteria.
25	Multiple	Multiple records must meet the criteria.
	Two	Two or more records must meet the criteria.
	Three	Three or more records must meet the criteria.
	Total	The total of all records must meet the criteria.
	Average	The average of all non-missing values must meet the criteria.
30	Count	A certain number of records must meet a specific criteria.

Domains

Domains allow for further constraints on the relationships with many transactions to one customer. For each of the above patterns of relationships, domains
5 allow for the restriction of records to a certain subset of the actual records that are subject to the pattern matching. For example, suppose the user wanted to know how many customers had three purchases of \$25 or more during 1992. A domain could be defined that would limit
10 transactions to 1992. The following query would then get the answer to this question:

Three_purchase.order_amount over 25

In this example, the domain would limit transactions to the year of 1992 and the query would
15 count those customers who had at least 3 purchases over \$25.

Universe Maps

Users frequently use the same query on many
20 different occasions. Database Link can save these queries as a universe and save the cost of creating the bitmap. Universe maps can be an individual query of the complex combination of several different queries put together.

25

Query Processing

The processing of the query is handled in a sequential fashion and is shown in Table 6.

TABLE 6: Query Processing

5	<p>Step 1 The server receives the query from the user as an ASCII character string. All queries are placed in a buffer where they are accessed as the query engine is available for processing. Once removed from the queueing buffer, the query is parsed into a series of basic queries. For example, the query:</p> <p style="text-align: center;">Gender is male and State is Minnesota</p> <p>would be parsed into two basic queries with an and operator connecting the two together.</p>
	<p>Step 2 The individual pieces of the query are placed on a decision tree with each branch node containing either a basic query or an operation. The above example would contain two branches with a basic query element at the end of each branch. More complex queries containing parentheses and various combinations of ands and ors would be broken down into more complex branch structures representing the proper order of precedence.</p>
	<p>Step 3 DBLink processes each of the queries at the end of each of the nodes. The result of this processing is a series of bitmaps that represent the results of each of these queries.</p>
	<p>Step 4 DBLink proceeds to combine the bitmaps together based on the operators that reside at each of the junctions between the individual queries. Once this process has been completed, an overall bitmap representing all of the table elements that meet a certain criteria has been created. The bits that are set on this map are then counted and the result is returned to the client.</p>

10

Post Query Processing

After the queries have been processed and a
 15 final result has been created, it is typical that additional information needs to be gathered about the customers that have been selected and counted. Reports and other information can be generated by Database Link.

20

Activity/Line Items Spec

Marketing databases often contain a complex
 relationship between the customer level data, activity
 level data, and product/line item level of data. Within
 25 the Database Link Database, there is a many-to-one relationship between customer level and activity level

records. Additionally, there is a many-to-one relationship between activity records and line item records.

5 Query Level

One of the most complex issues in dealing with queries at the purchase and line item level is the level of outcome for the query. The level of outcome is whether or not a resultant count is in customers, activity transactions, or line items. The level of outcome is totally separate from what information is used to compute the result. Thus, customer level results may use purchase and line item level information while line item level results may be affected by purchase level information. There are essentially three levels of queries across the Database Link system:

- **Customer level queries.** Customer level queries are by far the most common type of queries in Database Link. The result of a customer query is the number of customers that meet a certain criteria. The most common type of customer level query is the query that is asked at the customer level. For example, "gender is male" will give you an answer that says how many customers are males.

One may also want to answer queries at the customer level that are based on information at the activity or line item level. For example, one may want to know how many customers bought over \$50 worth of line item in 1992. This requires information from the purchase and line item level to be summarized into an answer regarding the number of customers. These types of queries all begin with some type of prefix that tells Database Link to take this query to the customer level.

- **Activity Level Queries.** Activity level queries give results that are at the level of individual

transactions. A typical question might be: how many purchases were made in 1992 from the Teens and Tots Catalog. The resultant answer would be the number of transactions that met all of these criteria. This answer will not say how many customers made these transactions, only the absolute number of transactions that were made at this criteria.

- **Line Item level Queries.** Line item level queries give results that are at the level of individual line items within a particular activity. A typical question might be: How many 486 computers were sold in the first four months of 1993. The resultant answer would be the number of items that contained a 486 computer.

Query Domains

Query domains are parts of queries that restrict the range of records that are used by another part of the same query. They have little meaning by themselves, but provide "staging" information for another part of the query. Query domains can also be applied at any of the three levels that queries can be constructed. Examples of the types of questions that are answered using domains are as follows:

1. *How many customers bought over \$100 in 1992?* In this case the domain is all activity transactions that occurred in 1992.
2. *How many purchases were made in 1992 from the Teens and Tots catalog that included teen clothing?* The domain for this question is the activity transactions that included a teen clothing line item.

Activity Queries

- **The Query Type.** The query type defines the pattern of relationship that is to be selected from amongst the activity records for a particular customer.
5 Examples of query types include first activity, second activity, or last activity. The result of this query is the number of customers who have activity records that meet the specific criteria.
Some query types are unique in that they refer to one specific record. The above examples all refer to a decision that can be made on one record. Other activity queries such as last 5 activities look to see if any one of the past five activities meet the criteria of the query.
10
- **Special Types.** All of the query types that are defined above work in a similar manner. There are three types of special queries:
Total_activity: The total activity query looks at the total amount that follows. An example:
20 Total_activity.amount over \$50.
Average_activity: The average activity query looks at the average amount that follows. An example:
Average_activity.amount under \$20.
Activity_count: The activity_count query is unique in that it does not have any field name that follows. It only counts the number of activity records that meet the criteria. An example:
25 Activity_count over 2.
- **The Activity Domain.** The activity domain defines the domain of activity records that are to be included in the query. This domain defines the "aperture" that frames those activity records that are to be included in the particular query type.
30
35 An activity domain consists of a series of activity level queries that are combined together to define a bitmap of activity records that are to be

considered for the particular query type. For example, an activity domain could be defined to be 1992 purchases. The activity domain could be named 1992_purchases and be defined as

5 *activity.order_date during 1992 and activity.amount*
over \$0. The full query would be
first_activity.amount over \$50 and activity_domain
is 1992_purchases. This query would select all
customers whose first purchase in 1992 was over
10 \$50.

All activity domains are defined as a function of pure activity functions. For example, a domain cannot be defined to be *first_activity.amount over \$0*.

15 • **The Line Item Domain.** The activity domain defines the domain of activity records that are to be included in the query. In the case of an activity query, the line item domain also restricts the domain of activity records that are considered.
20 The line item queries are first combined into one result and activity records are marked that have **any** records that are marked. One could then define the following domain as *large_shoes: line item.code is shoes and line item.size gt 10*. The activity
25 query: *first_activity.amount over \$50 and activity_domain is 1992_purchases and line item domain is large_shoes* would select those customers who had a first purchase over \$50 in 1992 when that order included large shoes.

30 • **Multiple Domains.** Domains can be combined together to form very complex selections of activity records and in turn customers. One could define a query to be: *last_activity.amount over \$50 and activity_domain is 1992_purchases and*
35 *activity_domain is 1993_purchases and line item_domain is large shoes.*

Line Item Queries

Line item queries are much more limited in scope than purchase queries.

- 5 • **The Query Type.** The only standard query type is any_line query. This query groups all line items together into one batch and performs a simple query. This would be identical to making any_activity query with a specific line item domain. However, because the activity level
10 definition is not needed, using the any_line item definition would be considerably more efficient.
- **Special Types.** All of the query types that are defined above work in a similar manner. There are three types of special queries:
15 **Total_line item:** The total activity query looks at the total amount that follows. An example:
Total_line item.amount over \$50.
 Average_line item: The average activity query looks at the average amount that follows. An
20 example: Average_line item.amount under \$20.
 Line item_count: The activity_count query is unique in that it does not have any field name that follows. It only counts the number of activity records that meet the criteria. An example
25 Purchase_count over 2.

All line item queries can be followed by any combination of purchase and line item domains. The only difference is that purchase level domains must
30 be expanded to the line item level rather than having line item level data reduced down to the purchase level. In this case all of the line item level records that meet the criteria of the purchase domain are marked. For example:
35 Total_line item.amount over \$50 and line item_domain is large_shoes and purchase_domain is

1992_purchases would select all customers who bought over \$50 worth of large_shoes in 1992.

Database Link Reports

5 The Database Link reporting engine is the direct complement to the other part of the Database Link program: the query engine. The query engine's primary task is to identify which individuals are to be examined. Each individual that is to receive special
10 attention has a bit set that marks the record for special consideration. The reporting engine accumulates some type of information about those individuals that have been marked.

 The basic idea of the report engine is to
15 accumulate counts or totals into an n-way data matrix. Each of the dimensions of the data matrix is defined by one of the dimension objects that are allocated off of the report structure.

 The object-oriented approach allows for a wide
20 variety of reports to be built around a few core data structures and functions.

The Report Structure

 The report structure contains all of the data
25 and program code to execute a particular type of report. A report is of a particular type. A report consists of a series of dimensions (up to five).

```
typedef struct{
    int          report_type;
    30      int          n_of_dim;
        DIMENSIONS    *dim[5];
        BITMAPS        *reference[5];
        int          joining_tables;
        MATRIX        counts;
    35      MATRIX        profits;
        MATRIX        orders;
        MATRIX        sales;
        }REPORT;
```

The Dimension

A dimension stores all data and processes relating to a particular dimension in a report.

Currently, Database Link is set up to process reports
 5 for up to four different dimensions. A dimension consists of several basic pieces of information: a field in the database, a pointer to a column of data, a set of labels, and a look-up function together with associated working structures. The current definition
 10 of the dimension structure is as follows.

```
typedef struct DIMENSIONS{
    FIELDS                                *xfields;
    unsigned char                        *data;
    unsigned char                        *start_address;
  15  int                                channel;
    int                                (*lookup_fxn) ();
    char                                table_name[30];
    TABLE                              *hash_table;
    unsigned short                      *lookup_map;
  20  unsigned char                      **lookup_map2;
    LABELS                             *labels;
    int                                current_label;
    int                                max_labels;
    int                                start_pos;
  25  int                                bytes_to_move;
}DIMENSIONS;
```

Table 4

30 In the above structure, the first element points to a structure called `xfields`. This substructure contains the information relevant to the field that is being used for a particular dimension.

The next several lines contain information
 35 about the actual column of data to be processed by this particular report. The `*data` is a pointer to a particular element of data within a column of data for a particular record of data. The `*start_address` is the address of the beginning of the data column. This is
 40 the point where the column begins, independent of where processing might be at a particular point in time. This data element is used for reports that must be reset and reprocessed several times.

The next several lines within the dimension structure are used to take a data point and perform a lookup function that returns an index into a set of data arrays. The pointer `*lookup_fn()` is a pointer to a
 5 function that is unique to the particular data type that is specified in the `xfields` data structure. The various lookup functions are specified in the following section. `Hash_table`, `lookup_map`, and `lookup_map2` are data structures used by the various lookup functions.

10 The last block of elements within the dimension structure relates to the labels that are used along that particular dimension. For example, in the case of a dimension based on gender, the labels would consist of male, female, and unknown. The `labels` data
 15 structure contains an array of label structures, each consisting of the following elements:

```
typedef struct{
    char          *banner;
    int           position;
    20    int         low_level;
    int         high_level;
    char        char_value[10];
    }LABELS;
```

25 Table 5

The `banner` is the actual label to be printed on the report while the other elements are used for the actual creation of a report.

30 The `current_label` element on the dimension structure is an index to the next available label. This is used in situations where the values are not known ahead of time and the list of labels is being filed in a dynamic fashion. The `max_labels` is an element that is
 35 the total number of labels that are used for a particular dimension.

The Lookup Routines

The lookup routines are the critical functions that take an incoming data point and use the data stored on the dimension structure to determine an index value to the data matrices for a report. There are several types of data structures, each optimized for a particular type of data and whether or not the values for a dimension are known ahead of time.

10 Character Fields

Character fields use an unsigned character and thus have a maximum of 256 unique values. Characters can thus be mapped to a particular index by using an allocated array of 256 values. For each data point that needs to be evaluated, the data value of the unsigned character can be used as an index into an array of shorts that contains the value to be assigned. This represents the most efficient method of performing a lookup function.

20 Small Integer Fields

Small integers are stored in one byte fields and thus are mapped in a similar fashion to single character data fields.

Medium Integer/Date Fields/Fixed Strings

25 Medium integers, dates, and fixed strings are stored as 16-bit integers and thus have a maximum value of 65535. These fields are also dealt with as a single lookup operation on an array of shorts that is 65536 elements long. Each element contains the index value to the data matrix.

30 Large Integer/Dollar Fields

Because of the very large number of possibilities for a large integer, it is not possible to use a simple map function to put individual values to particular index values. However, because most integers (dollars in particular) are between 0 and 20,000, it is possible to directly map these smaller values to an

index while using more computer intensive methods only for the larger values.

If the value exceeds the length of the map (20,000), then these alternative methods are used. If pre-defined ranges are in effect, then a simple loop is used to evaluate each of the ranges. If no ranges are in effect, then a hashing system similar to that used in character strings is used to efficiently map the values to a particular output.

10 Character Strings

Character strings are the most inefficient data type to process. The lookup function for characters uses the same hashing function that is used in the load program. The character string is hashed into an array of pointers to linked lists. Once the top of the list is accessed, the lookup function traverses the list until a match is found. If no match is found, then a new link is added at the end of the list.

A special case of character strings is the field that contains exactly two characters. Several commonly used strings such as state and country codes use two character fields for storage of data values. Because valid ASCII character values must be between 1 and 128, it is possible to build a 2 dimensional 128 by 128 matrix (16384 total elements) that stores the index value for each possible combination of the two characters. This allows for using the two characters as indices into this array that contains the index into the data matrix.

30

Report Processing

Once the report and dimensional structures have been populated, the processing of reports proceeds. This is always a two-part process: 1) a bitmap is set that determines which cases will be included in a report, then 2) the query engine proceeds to fill the data matrix with accumulated data. The following is an

outline in pseudo-code for the creation of simple two-way cross-tabs reports:

```

    setup report two-way matrix
    setup row dimension
5   setup column dimension
    for each case in the table
        is current bit in bitmap set
            row = lookup of row data
            column = lookup of column data
10        report.matrix[row] [column] increment by
            1
        increment row data
        increment column data
        increment to next bit in bitmap
15
```

Once all of the set up has taken place, the actual processing is quite simple. Prior to the actual generation of the report, a query has been executed that
20 produces a resultant bitmap that marks every "record" in the table that should be included in the report. During the setup of each dimension, a pointer to the top of the data column for that particular dimension is created. In the case of a two-way crosstabs, this involves the
25 setting of two pointers--one for the row variable and one for the column variable. The dimension structures also are set to point to the appropriate lookup functions.

The most common application of this is for tracking
30 data on multiple purchases for one individual. With this type of many-to-one relationship, there are many possible patterns of relationships that can be formed. We have developed specific retrieval functions that allow us to get at data relating to the first and last
35 purchase that a customer has made in a particular mode.

The invention is further described in Appendices A, B, C, and D.

5

10

Appendix A for U.S. Patent Application

15

DATABASE LINK SYSTEM
BACKGROUND OF THE INVENTION

Filed October 22, 1993

Appendix A: Main Routine for the FDC Prototype Executable

20

© 1993 FDC, Inc.
All Rights Reserved

25

30

35

```

/*
-- MODULE: FDC_PROTOTYPE.C
--
5  -- MODULE DESCRIPTION:
--
--    main routine for the FDC_PROTOTYPE executable.
--
--
-- AUTHORS:
--
--    Kelly Westman    Digital Equipment Corporation
--    Sushil Pillai    Digital Equipment Corporation
--
10  -- CREATION DATE: 30-July-1992
--
-- DESIGN ISSUES.
--
--
-- PORTABILITY ISSUES:
--
--    Assumes logical "master_file" is present. This defines the file which
--    describes the fields contained in the section file of data.
--
15  -- MODIFICATION HISTORY:
--
--    30-July-1992 - Original
--    11-May-1993 - Charles Malmeskog
--        - Removed fail_and_exit(), fail_and_exit_system();
--        - Replaced with error_handler(). - More graceful exits and retries
--        - Updated exit_normal(): to do some cleanup before the exit.
--    25-May-1993 - Charles Malmeskog
--        - Line up and center some Header information, pull out error_mesg_routine
20  --    15-June-1993 - Charles Malmeskog
--        - Error handler should clear out all the message buffers before sending an
--        error message to the client. On the server side we will send the buffers
--        to the screen for debug use.
--        - Free up memory the insert_comma routine uses
--    18-June-1993 - Dan Thomas
--        - Add Customer to Purchase, Customer to Produce and Purchase to Product
--        flush and UnmapClosefile routines & related external data
--    21-June-1993 - Charles Malmeskog
--        - Center_string based on 70 characters
25  --    30-Sep-1993 - Charles Malmeskog
--        - Update error_handler() calls to include ERROR message type
*/

/*
-- INCLUDE FILES
*/

30  #include <stdio>
#include <unistd>
#include <file>
#include <math>
#include <sysdef>

/*
-- error_handler specific
*/
#include "fdc_error_numbers.h"

35  #include "fdc_parser.h"
#include "fdc_prototype.h"
#include "hash.h"

```

```

/*
 * DMO specific
 */
#include "fdc_dmq_struct.h"
#include "fdc_dmq_types.h"
#include "dmq$user.p_symbol.h"

#define CIS_EOI 10
#define DBG TRUE

char      *NullString = "";

int      DMO_CONNECTED;
pams_addr target;
int      msg_offset;
char      msg_put_buffer[MAX_BUFFERS];
char      msg_file_buffer[FILE_BUFFER_LENGTH];
char      msg_summary_buffer[BUFFER_LENGTH];
short     pams_get_msg_type;
short     pams_get_msg_class;
short     pams_put_msg_type=MSG_SINGLE_TYPE;
short     pams_put_msg_class=PAMS_MSG_CLAS;

extern char *msg_get_buffer;
char      msg_buffer[BUFFER_TERM_LENGTH];
char      *msg_header;
char      *pams_message;

ntwk_hdr_t *ntwk_hdr;
int      read_ntwk_hdr = TRUE;
int      ntwk_pkt_type;
int      packet_processing = 0;
char      *query_keycode = "";

char      *query_label = ""
char      query_filename[7]
char      db_name[FILE_NAME_LENGTH]
char      db_version_date[31];
int      set_subsidiary;
int      set_purchase;
int      set_product;
char      *subsidiary_labels(SUB_FILE_MAX);
int      query_num=0;
char      customized_query[128];
char      *conditional_query;

unsigned int reset_segmentation;
unsigned char segmentation_response;

/* map flushing flags. 0=no flushing map count data. 1=openmapfile and unmapclose maps with ea query */
unsigned int delete_pur_info = 0; /* flush map count flag. customer to purchase map counts */
unsigned int delete_prd_info = 0; /* flush map count flag. customer to product map counts */
unsigned int delete_pur_prd_info = 0; /* flush map count flag. purchase to product map counts */

/*
 * Distribution specific
 */

```

```

#define GET_FIELD_NAME() { \
    atsign = strchr(pams_message, '@'); \
    switch (j) \
    { \
        case 0 : memcpy (dist_info[i].field_name, pams_message, atsign - pams_message); \
        dist_info[i].field_name[atsign-pams_message] = '\0'; \
        break; \
        case 1 : memcpy (xtab_info[i].hz_field_name, pams_message, atsign - pams_message); \
        xtab_info[i].hz_field_name[atsign-pams_message] = '\0'; \
        break; \
        case 2 : memcpy (xtab_info[i].vt_field_name, pams_message, atsign - pams_message); \
        xtab_info[i].vt_field_name[atsign-pams_message] = '\0'; \
        break; \
        case 3 : memcpy (xtab_info[i].pg_field_name, pams_message, atsign - pams_message); \
        xtab_info[i].pg_field_name[atsign-pams_message] = '\0'; \
        } \
    pams_message = atsign + 1; \
}

#define GET_TABLE_NAME() { \
    atsign = strchr(pams_message, '@'); \
    switch (j) \
    { \
        case 0 : memcpy (dist_info[i].table_name, pams_message, atsign - pams_message); \
        dist_info[i].table_name[atsign-pams_message] = '\0'; \
        break; \
        case 1 : memcpy (xtab_info[i].hz_table_name, pams_message, atsign - pams_message); \
        xtab_info[i].hz_table_name[atsign-pams_message] = '\0'; \
        break; \
        case 2 : memcpy (xtab_info[i].vt_table_name, pams_message, atsign - pams_message); \
        xtab_info[i].vt_table_name[atsign-pams_message] = '\0'; \
        break; \
        case 3 : memcpy (xtab_info[i].pg_table_name, pams_message, atsign - pams_message); \
        xtab_info[i].pg_table_name[atsign-pams_message] = '\0'; \
        break; \
        } \
    pams_message = atsign + 1; \
}

#define GET_MIN_VALUE() { \
    switch (j) \
    { \
        case 0 : dist_info[i].min_value = atoi(pams_message); \
        break; \
        case 1 : xtab_info[i].hz_min = atoi(pams_message); \
        break; \
        case 2 : xtab_info[i].vt_min = atoi(pams_message); \
        break; \
        case 3 : xtab_info[i].pg_min = atoi(pams_message); \
        } \
    pams_message = strchr(pams_message, '@') + 1; \
}

#define GET_MAX_VALUE() { \
    switch (j) \
    { \
        case 0 : dist_info[i].max_value = atoi(pams_message); \
        break; \
        case 1 : xtab_info[i].hz_max = atoi(pams_message); \
        break; \
        case 2 : xtab_info[i].vt_max = atoi(pams_message); \
        break; \
        case 3 : xtab_info[i].pg_max = atoi(pams_message); \
        break; \
    } \
}

```



```

    }
    pams_message = strchr(pams_message, '@') + 1;
}

5      xtab_info_t xtab_info[10];
      int xtab_info_idx;
      dist_info_t dist_info[10];
      int dist_info_idx;
      int num_of_dist;
      char *dist_start_pos;

      /*
      - Config file specific.
10 */

      #define MAX_TOKEN_NAME_LEN 26
      #define MAX_TOKEN_VALUE_LEN 40

      char config_file[80] = "admin_dir/config.dat";
      handle ht; /* handle to a hash table */

      /*
      - Not sure what specific.
15 */

      int max_number_of_bits = 0;
      int max_product_bits = 0;
      int max_purchase_bits = 0;
      int row_count = 1;
      int bit_counts[256];
      int initial_occurrence=TRUE;

20 /*
      - Debug variables.
      */

      int parser_debug;
      FILE *debug_file;

      /*
      - External declarations
25 */

      extern struct bitmap *super_master_bitmap;
      extern int msg_outstanding;
      extern int msg_not_outstanding;

      /* flush variables */
      extern int cus_pur_chnl /* channel - customer-purchase map count for OPENMAPFILE */
      extern int cus_pro_chnl /* channel - customer-product map count for OPENMAPFILE */
      extern int pur_pro_chnl /* channel - purchase-product map count for OPENMAPFILE */
30 extern struct address_range cus_pur_addr /* map addresses - customer-purchase map count for
      OPENMAPFILE.UNMAPCLOSE */
      extern struct address_range cus_pro_addr /* map addresses - customer-product map count for
      OPENMAPFILE.UNMAPCLOSE */
      extern struct address_range pur_pro_addr /* map addresses - purchase-product map count for
      OPENMAPFILE.UNMAPCLOSE */

      extern int cus_pur_map /* 0=not mapped, 1=mapped, customer to purchase map count data */
      extern int cus_pro_map /* 0=not mapped, 1=mapped, customer to product map count data */
35 extern int pur_pro_map /* 0=not mapped, 1=mapped, purchase to product map count data */

      /* header flag information */
      short int header_flags[NUMBER_OF_FLAGS];

```

```

/*
 * Function prototypes for this module.
 */

5 void null_terminate();
  void load_config_file();
  init_dmq_buffers();
  free_dmq_buffers();
  set_dmq_buffer();
  int upper_strcmp();
  exit_normal();
  long check_ret();

10 /*
   * Main routine. Loads field definitions from MASTER_FILE. initializes.
   * Then prompts for entry. Entry is parsed and dealt with. If EXIT command
   * the history of entries and results is displayed and program exits.
   * If query command, the query is resolved, and the results put in the parse
   * log, and the parse tables are reset for the next entry. If the UEND.
   * Universe End, the universe history is display and the parse tables are
   * reset. If sample end then endsampling routine is called.
15 */

main(argc, argv)
int argc;
char *argv[];
{
    int i, j, len, status, num_tables;
    long int attach_mode, q_type;
    char q_name[QUEUE_NAME_LENGTH];
    long int q_name_len;
20 pams_addr new_process_num;
    char *s1;
    char *atrig;
    char *save_pams_msg;
    char segmentation_char;
    void initialize_variables();
    char temp_string15[15];
    char *temp_string_pointer;
25 char temp_string85[85];

    extern struct expression *current_expr;

    int pkt_debug = FALSE;
    int pkt_simdbg = FALSE;

    /*
     * Initialization.
30 */

    DMQ_CONNECTED = FALSE;
    packet_processing = FALSE;

    pams_get_msg_type = MSG_SINGLE_TYPE;

    /*
     * Process command line arguments
35 */

```

```

for (i = 1; i < argc; i++)
{
    s1 = strtoupper (argv[i]);

    if (strcmp(s1, "DMQ") == 0)
        DMO_CONNECTED = TRUE;
    else if (strcmp(s1, "PKT") == 0)
        packet_processing = TRUE;
    else if (strcmp(s1, "PKTDBG") == 0)
    {
        pkt_debug = packet_processing = TRUE;
        printf ("debugging network packets.\n");
    }
    else if (strcmp(s1, "PKTSIM") == 0)
    {
        pkt_simdbg = pkt_debug = packet_processing = TRUE;
        printf ("simulating network packets (use ^ to separate packets)\n");
    }
    else if (strcmp(s1, "PARSERDBG") == 0)
    {
        parser_debug = TRUE;
        printf ("debugging parser\n");
    }
    else if (strcmp(s1, "PUF") == 0) /* flush flag - customer to purchase map counts */
    {
        delete_pur_info = 1;
    }
    else if (strcmp(s1, "PRF") == 0) /* flush flag - customer to product map counts */
    {
        delete_prd_info = 1;
    }
    else if (strcmp(s1, "PPF") == 0) /* flush flag - purchase to product map counts */
    {
        delete_pur_prd_info = 1;
    }
    else
        printf ("Invalid argument %s ignored\n", s1);

    free (s1);
    s1 = NULL;
}

#ifdef DBG
    debug_file = fopen("debug_file" "w");
#endif

/*
-- Read files
*/
load_config_file (config_file);

load_fields ("master_file", &max_number_of_brs, db_name, db_version_date, &set_subsidary,
             &set_purchase, &set_product, subsidiary_labels);
set_counter();
if ((set_subsidary) || (set_purchase) || (set_product))
{
    load_bitmaps();
    if ((set_purchase) || (set_product))
    {
        init_pur_prd_table();
        load_pur_prd_maps();
    }
}

```

```

/*
 * Initialize network and I/O buffers.
 */

5  set_dmq_buffer();
   memset(msg_file_buffer, 0, FILE_BUFFER_LENGTH);
   init_dmq_buffers();
   if (DMQ_CONNECTED)
   {
       attach_mode = PSYM_ATTACH_BY_NUMBER;
       q_type = PSYM_ATTACH_PQ;

       strcpy(q_name, "1");
       q_name_len = strlen(q_name);
10  status = pams_attach_q (&attach_mode, &new_process_num.all, &q_type,
                           q_name, &q_name_len, 0, 0, 0, 0, 0);

       if (!check_ret(status))
       {
           error_handler (DMQ_ATTACH_FAIL, ERROR, status, NO_STRING);
       }
       load_config_file (config_file);
   }

15

   /*----- TEMPORARY!!! -----*/

   if (!packet_processing || pkt_simdbg)
   {
       ntwk_hdr = (ntwk_hdr_t *) malloc (NTWK_HDR_SIZE);
       CHECK_ALLOCATION(ntwk_hdr, "ntwk_hdr Routine main()");

20  strcpy (ntwk_hdr->dbname, "BNBL");
       memcpy (ntwk_hdr->query_id, "001", 3);
       ntwk_hdr->query_id[3] = '\0';
       memcpy (ntwk_hdr->subdate, "01/01/1992", 10);
       ntwk_hdr->subdate[10] = '\0';
       memcpy (ntwk_hdr->subtime, "00 00", 5);
       ntwk_hdr->subtime[5] = '\0';
       memcpy (ntwk_hdr->runtime, "00 00", 5);
       ntwk_hdr->runtime[5] = '\0';
       strcpy (ntwk_hdr->label, "A label of some sort");

25  strcpy (query_filename, ntwk_hdr->query_id, 3);
       query_filename[3] = '\0';
   }

/*
 * Process each of the packets. If connect to DMQ.
 * remain connected upon CIS_EOI or EXIT status
 */
30 while (TRUE)
   {
       /*
        * Read the header packet (along with all non-query packets)
        */

       if (read_ntwk_hdr)
       {
           /*
            * Initialization every header read.
            */
35

```

```

    read_nwtk_hdr = FALSE;

    /*
    ~ Perform the I/O
    */
5
    if (DMQ_CONNECTED)
    {
        dmq_get_msg();
        pams_message = msg_get_buffer;

        if (pkt_debug)
            printf ("msg from nwtk: %s\n", pams_message);
10
        }
        else
        {
            msg_buffer[0] = '\0';
            printf ("Cmd> ");
            gets(msg_buffer);
            pams_message = msg_buffer;
        }

15
    if ((!msg_outstanding) && (!msg_not_outstanding))
    {
        /*
        ~ Strip preceding spaces and newlines...
        */
        while ((*pams_message <= ' ') && (*pams_message != '\0') &&
            (*pams_message != PACKET_SEPERATOR)) pams_message++;
        /*
        ~ Make sure there is data there!
        */
20
        if (*pams_message == '\0')
        {
            read_nwtk_hdr = TRUE;
            continue;
        }

        /*
        ~ Check for a quick exit command!
        */
25
        if (strcmp(pams_message, "exit", 4) == 0)
        {
            error_handler(ILLEGAL_EXIT_STRING, ERROR, NO_STATUS, NO_STRING);
            continue;
        }

30
        /*
        ~ Process the header packets from the windows app.
        */

        if (packet_processing && DMO_CONNECTED || pkt_debug)
        {
            if (pkt_simdbg)
            {
                SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 1);
                printf ("SEPERATOR\n");
            }
            else
            {
35

```

```

5      SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 1).
      if (pkt_debug)
        printf ("SEPERATOR\n");
    }

    /*
    -- Process the header packet!
    */

    if (!pkt_simdbg)
    {
10        ntwk_hdr = (ntwk_hdr_t *) pams_message.

        /*
        -- Terminate the variable length strings with a '\0'.
        */

        null_terminate (ntwk_hdr->dbname, NTWK_HDR_DBNAME_LEN).
        null_terminate (ntwk_hdr->header_flags, NUMBER_OF_FLAGS).
        null_terminate (ntwk_hdr->label, NTWK_HDR_LABEL_LEN).

15        /*
        -- Process header flags
        */

        if (ntwk_hdr->header_flags[VER_TOTAL] == '0')      /* Vertical Totals */
            header_flags[VER_TOTAL] = NO_TOTALS.
        else if (ntwk_hdr->header_flags[VER_TOTAL] == '1')
            header_flags[VER_TOTAL] = SUM_TOTALS.
        else if (ntwk_hdr->header_flags[VER_TOTAL] == '2')
            header_flags[VER_TOTAL] = AVG_TOTALS.
20        else if (ntwk_hdr->header_flags[HOR_TOTAL] == '3')
            header_flags[HOR_TOTAL] = BOTH_TOTALS.
        else
            header_flags[HOR_TOTAL] = -1; /* default */

        if (ntwk_hdr->header_flags[HOR_TOTAL] == '0')      /* Horizontal Totals */
            header_flags[HOR_TOTAL] = NO_TOTALS.
        else if (ntwk_hdr->header_flags[HOR_TOTAL] == '1')
            header_flags[HOR_TOTAL] = SUM_TOTALS.
25        else if (ntwk_hdr->header_flags[HOR_TOTAL] == '2')
            header_flags[HOR_TOTAL] = AVG_TOTALS.
        else if (ntwk_hdr->header_flags[HOR_TOTAL] == '3')
            header_flags[HOR_TOTAL] = BOTH_TOTALS.
        else
            header_flags[HOR_TOTAL] = -1; /* default */

        printf ("header flag zero = %d\n", header_flags[0]); /* hard coded until windows */
        printf ("header flag one = %d\n", header_flags[1]); /* gives us correct inputs CM */
        printf ("set both total flags to on\n"); /* hard code till windows catch up CM */
30        header_flags[0] = 1; /* SUM */ /* hard code till windows catch up CM */
        header_flags[1] = 1; /* SUM */ /* hard code till windows catch up CM */
        printf ("header flag zero = %d\n", header_flags[0]); /* hard code till windows catch up CM */
        printf ("header flag one = %d\n", header_flags[1]); /* hard code till windows catch up CM */

        header_flags[2] = -1.
        header_flags[3] = -1. /* rest are just hard coded to -1 for now CM */
    }

```

35


```

header_flags[4] = -1; /* we will use some for reports later    CM */
header_flags[5] = -1;
header_flags[6] = -1;
header_flags[7] = -1;
5 header_flags[8] = -1;

/*
 * Build the query filename.
 */
strcpy (query_filename, ntwk_hdr->query_id, 3);
query_filename[3] = '\0';

if (pkt_debug)
10 printf ("read header. dbname = %s, query_filename = %s\n      label = %s\n",
        ntwk_hdr->dbname, query_filename, ntwk_hdr->label);

/*
 * Skip past the network header.
 */

pams_message += NTWK_HDR_SIZE;
} /*endif */

15 /* Build the header for every query */
msg_header = calloc(FILE_BUFFER_LENGTH, sizeof(*msg_header));
sprintf(temp_string15, "%d", max_number_of_bits);
temp_string_pointer = insert_comma(temp_string15);
sprintf(msg_header, "DATABASE : %s DB VERSION : %s\nTOTAL CUSTOMER RECORDS
%s\n", db_name, db_version_date, temp_string_pointer);
free (temp_string_pointer);
temp_string_pointer = NULL;

if (DMO_CONNECTED)
20 {
    msg_buf_xfer (msg_header);
}
else
{
    printf ("%s", msg_header);
}
free(msg_header);
msg_header = NULL;

if (pkt_s:mdbg)
25 {
    SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2);
    printf ("SEPERATOR\n");
}
else
{
    SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 2);
    if (pkt_debug
30 printf ("SEPERATOR\n");
}

/*
 * Read the packet type
 */
if (pkt_debug)
    printf ("packet type = %c%c\n", *pams_message, *(pams_message+1));

ntwk_pkt_type = PACKET_TYPE(pams_message);
pams_message += 2;
35

```

```

switch (nwkw_pkt_type)
{
    case PKT_QUICK_QUERY
    case PKT_COUNT_QUERY
5      /* Quick & Count Queries */
        break;

    case PKT_DISTRIBUTION
10      /* Display Distribution */

        num_tables = 1;
        if (pkt_simodb)
        {
            SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2);
            printf ("SEPERATOR\n");
        }
        else
15      {
            SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 2);
            if (pkt_debug)
                printf ("SEPERATOR\n");
        }

        num_of_dist = atoi(pams_message);
        if (pkt_debug)
            printf ("number of distributions is %d\n", num_of_dist);
20      pams_message = strchr(pams_message, '@') + 1;

        for (i = 0; i < num_of_dist; i++)
        {
            for (j=0; j<num_tables; j++)
            {
25              GET_FIELD_NAME();
              GET_TABLE_NAME();

              /* 4/17 svp - we have decided not to use min and max frequencies. This
              functionality may be used later and hence it is commented out */

              GET_MIN_VALUE();
              GET_MAX_VALUE();
            }

            if (pkt_debug)
30            {
                printf ("field=%s, table=%s, min=%d, max=%d\n",
                    dist_info[i] field_name, dist_info[i] table_name,
                    dist_info[i] min_value, dist_info[i] max_value);
            }

            /* svp - check on this, not being used. */
            dist_start_pos = pams_message;
            dist_info_idx = 0;
35            break;
        }
    }
}

```

```

case PKT_COND_DIST :
/*
5  - Display Conditional Distribution!
*/

    num_tables = 1;
    if (pkt_simdbg)
    {
        SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2);
        printf ("SEPERATOR\n");
    }
10  else
    {
        SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 2);
        if (pkt_debug)
            printf ("SEPERATOR\n");
    }

    num_of_dist = atoi(pams_message);

15  if (pkt_debug)
        printf ("number of distributions is %d\n", num_of_dist);

    pams_message = strchr(pams_message, '@') + 1;

    for (i = 0; i < num_of_dist; i++)
    {
        for (j=0; j<num_tables; j++)
        {
20          GET_FIELD_NAME();
          GET_TABLE_NAME();

          4/17 svp - we have decided not to use min and max frequencies. This
          functionality may be used later and hence it is commented out.

          GET_MIN_VALUE();
          GET_MAX_VALUE();

          }

25          if (pkt_debug)
          {
              printf ("field=%s table=%s min=%d max=%d\n",
dist_info[i] field_name dist_info[i] table_name,
dist_info[i] min_value dist_info[i] max_value);
          }

        }

30  /* svp - check on this, not being used */
    dist_start_pos = pams_message;
    dist_info_idx = 0;

    /* get the conditional query */
    atsign = strchr(pams_message, '@');
    len = (int)atsign - (int)pams_message;
    conditional_query = malloc(len+1);
    CHECK_ALLOCATION(conditional_query, "conditional_query Routine main()");

    memcpy (conditional_query, pams_message, atsign - pams_message);
35  conditional_query[len] = '\0';

```

```

    pams_message = strchr(pams_message, '@') + 1;
    break;
5   case PKT_COND_2WAY
    {
        /* 2-way Cross Tabs! */

        num_tables = 2;
        if (pkt_simdbg)
        {
            SEPERATOR_CHECK(pams_message++, DEBUG_SEPERATOR, 2);
            printf("SEPERATOR\n");
        }
        else
        {
            SEPERATOR_CHECK(pams_message++, PACKET_SEPERATOR, 2);
            if (pkt_debug)
                printf("SEPERATOR\n");
        }
10
        num_of_dist = atoi(pams_message);
        if (pkt_debug)
            printf("number of distributions is %d\n", num_of_dist);

        pams_message = strchr(pams_message, '@') + 1;
        for (i = 0; i < num_of_dist; i++)
        {
15
            /* for 2 way crosstabs, there are two sets of field names and table names */
            for (j=1; j<=num_tables; j++)
            {
                GET_FIELD_NAME();
                GET_TABLE_NAME();

                4/17 svp - we have decided not to use min and max frequencies. This
                functionality may be used later and hence it is commented out:

                GET_MIN_VALUE();
                GET_MAX_VALUE();
25
            }

            if (pkt_debug)
            {
                printf("v: field=%s v: table=%s, v: min=%d v: max=%d\n",
                    xtab_info[i] v: field_name, xtab_info[i] v: table_name,
                    xtab_info[i] v: min, xtab_info[i] v: max);

                printf("h: field=%s h: table=%s, h: min=%d h: max=%d\n",
                    xtab_info[i] h: field_name, xtab_info[i] h: table_name,
                    xtab_info[i] h: min, xtab_info[i] h: max);
30
            }
        }
    }

```

35

```

/* svp - check on this, not being used. */
dist_start_pos = pams_message;
xtab_info_idx = 0;

5  /* get the conditional query */
   atsign = strchr(pams_message, '@');
   len = (int)atsign - (int)pams_message;
   conditional_query = malloc(len+1);
   CHECK_ALLOCATION(conditional_query, "conditional_query Routine main()");

   memcpy(conditional_query, pams_message, atsign - pams_message);
   conditional_query[len] = '\0';

10  pams_message = strchr(pams_message, '@') + 1;

   break;

case PKT_COND_3WAY:
/*
-- 3-way Cross Tabs'
*/
   num_tables = 3;
   if (pkt_simdbg)
15  {
       SEPERATOR_CHECK(pams_message++, DEBUG_SEPERATOR, 2);
       printf("SEPERATOR\n");
   }
   else
   {
       SEPERATOR_CHECK(pams_message++, PACKET_SEPERATOR, 2);
       if (pkt_debug)
           printf("SEPERATOR\n");
20  }

   num_of_dist = atoi(pams_message);

   if (pkt_debug)
       printf("number of distributions is %d\n", num_of_dist);

   pams_message = strchr(pams_message, '@') + 1;

   for (i = 0; i < num_of_dist; i++)
25  {
       /* for 3 way crosstabs there are three sets of field names and table names */
       for (j=1; j<=num_tables; j++)
       {
           GET_FIELD_NAME(i);
           GET_TABLE_NAME(i);

           4/17 svp - we have decided not to use min and max frequencies. This
           functionality may be used later and hence it is commented out.

           30  GET_MIN_VALUE(i);
                GET_MAX_VALUE(i);

                }

                if (pkt_debug)
                {
                    printf(" v1_field=%s, v1_table=%s, v1_min=%d, v1_max=%d\n",
xtab_info[i].v1_field_name, xtab_info[i].v1_table_name,
xtab_info[i].v1_min, xtab_info[i].v1_max);

35  printf(" hz_field=%s, hz_table=%s, hz_min=%d, hz_max=%d\n",
xtab_info[i].hz_field_name, xtab_info[i].hz_table_name,
xtab_info[i].hz_min, xtab_info[i].hz_max);

```

```

                                printf (" pg_field=%s, pg_table=%s, pg_min=%d, pg_max=%d\n",
xstab_info[i].pg_field_name, xstab_info[i].pg_table_name,
xstab_info[i].pg_min, xstab_info[i].pg_max);
                                }
5      )

      /* svp - check on this, not being used. */
      dist_start_pos = pams_message;
      xstab_info_idx = 0;

      /* get the conditional query */
      atsign = strchr(pams_message, '@');
      len = (int)atsign - (int)pams_message;
10     conditional_query = malloc(len+1);
      CHECK_ALLOCATION(conditional_query, "conditional_query: Routine main()");

      memcpy (conditional_query, pams_message, atsign - pams_message);
      conditional_query[len] = '\0';

      pams_message = strchr(pams_message, '@') + 1;

      break;

15     case PKT_DISPLAY:
      /*
       * Display
       */

      num_tables = 1;
      if (pkt_smdbg)
      {
          SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2);
          printf ("SEPERATOR\n");
20      }
      else
      {
          SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 2);
          if (pkt_debug)
              printf ("SEPERATOR\n");
      }

      num_of_dist = atoi(pams_message);

25     if (pkt_debug)
        printf ("number of distributions is %d\n", num_of_dist);

      pams_message = strchr(pams_message, '@') + 1;

      for (i = 0, i < num_of_dist, i++)
      {
          for (j=0, j<num_tables, j++)
          {
30              GET_FIELD_NAME();
          }

          if (pkt_debug)
          {
              printf (" field=%s\n", dist_info[i] field_name)
          }
      }

35     /* svp - check on this, not being used */
      dist_start_pos = pams_message;
      dist_info_idx = 0;

      break;

```



```

case PKT_WRITE .
/*
 * Write!
 */
5
    num_tables = 1.
    if (pkt_simdbg)
    {
        SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2).
        printf ("SEPERATOR\n");
    }
    else
    {
10
        SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 2).
        if (pkt_debug)
            printf ("SEPERATOR\n");
    }

    num_of_dist = atoi(pams_message);

    if (pkt_debug)

15
        printf ("number of distributions is %d\n", num_of_dist);

    pams_message = strchr(pams_message, '@') + 1;

    for (i = 0; i < num_of_dist; i++)
    {
        for (j=0; j<num_tables; j++)
        {
            GET_FIELD_NAME().
        }

20
        if (pkt_debug)
        {
            printf (" field=%s\n", dist_info[i] field_name);
        }
    }

    /* svp - check on this, not being used */
    dist_start_pos = pams_message;
    dist_info_max = 0;

25
    break;

case PKT_ZWAY_XTABS
/*
 * 2-way Cross Tabs
 */

    num_tables = 2.
    if (pkt_simdbg)
30
    {
        SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2).
        printf ("SEPERATOR\n");
    }
    else
    {
        SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 2).
        if (pkt_debug)
            printf ("SEPERATOR\n");
    }

35
    num_of_dist = atoi(pams_message);

    if (pkt_debug)
        printf ("number of distributions is %d\n", num_of_dist);

```

```

pams_message = strchr(pams_message, '@') + 1;

for (i = 0; i < num_of_dist; i++)
{
    /* for 2 way crosstabs, there are two sets of field names and table names */
    for (j=1; j<=num_tables; j++)
    {
        5      GET_FIELD_NAME();
        GET_TABLE_NAME();

        4/17 svp - we have decided not to use min and max frequencies. This
        functionality may be used later and hence it is commented out.

        GET_MIN_VALUE();
        GET_MAX_VALUE();

10      }

        if (pkt_debug)
        {
            printf (" vt_field=%s, vt_table=%s, vt_min=%d, vt_max=%d\n",
xtab_info[i].vt_field_name, xtab_info[i].vt_table_name,
xtab_info[i].vt_min, xtab_info[i].vt_max);

            printf (" hz_field=%s, hz_table=%s, hz_min=%d, hz_max=%d\n",
15      xtab_info[i].hz_field_name, xtab_info[i].hz_table_name,
xtab_info[i].hz_min, xtab_info[i].hz_max);
        }
    }

    /* svp - check on this, not being used. */
    dist_start_pos = pams_message;
    xtab_info_idx = 0;

20      break;

case PKT_3WAY_XTABS
{
    /*
    == 3-way Cross Tabs!
    */
    num_tables = 3;
    if (pkt_simdbg)
    {
        25      SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2);
        printf ("SEPERATOR\n");
    }
    else
    {
        SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 2);
        if (pkt_debug)
            printf ("SEPERATOR\n");
    }
}

    num_of_dist = atoi(pams_message);

30      if (pkt_debug)
        printf ("number of distributions is %d\n", num_of_dist);

    pams_message = strchr(pams_message, '@') + 1;

    for (i = 0; i < num_of_dist; i++)
    {
        /* for 3 way crosstabs, there are three sets of field names and table names */
        for (j=1; j<=num_tables; j++)
        {
            35      GET_FIELD_NAME();
            GET_TABLE_NAME();

```

```

4/17 svp - we have decided not to use min and max frequencies. This
functionality may be used later and hence it is commented out.

    GET_MIN_VALUE();
    GET_MAX_VALUE();

5      }

      if (pkt_debug)
      {
          printf (" vt_field=%s, vt_table=%s, vt_min=%d, vt_max=%d\n",
xstab_info[i].vt_field_name, xstab_info[i].vt_table_name,
xstab_info[i].vt_min, xstab_info[i].vt_max);

          printf (" hz_field=%s, hz_table=%s, hz_min=%d, hz_max=%d\n",
10  xstab_info[i].hz_field_name, xstab_info[i].hz_table_name,
xstab_info[i].hz_min, xstab_info[i].hz_max);

          printf (" pg_field=%s, pg_table=%s, pg_min=%d, pg_max=%d\n",
xstab_info[i].pg_field_name, xstab_info[i].pg_table_name,
xstab_info[i].pg_min, xstab_info[i].pg_max);
      }

      /* svp - check on this not being used. */
15  dist_start_pos = pams_message;
xstab_info_idx = 0;

      break;

case PKT_RFM
/*
RFM'
*/
20  num_tables = 3;
if (pkt_simdbg)
{
    SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2);
    printf ("SEPERATOR\n");
}
else
{
    SEPERATOR_CHECK (pams_message++ PACKET_SEPERATOR, 2)
if (pkt_debug)
25  printf ("SEPERATOR\n");
}

num_of_dist = atoi(pams_message);

if (pkt_debug)
    printf ("number of distributions is %d\n", num_of_dist);

pams_message = strchr(pams_message, '@') + 1;

30  for (i = 0, i < num_of_dist, i++)
    {
        /* for 3 way crosstabs, there are three sets of field names and table names */
        for (j = 1, j <= num_tables, j++)
        {
            GET_FIELD_NAME();
            GET_TABLE_NAME();

            3/26 svp - we have decided not to use min and max frequencies. This
            functionality may be used later and hence it is commented out.

            GET_MIN_VALUE();
            GET_MAX_VALUE();
35

```

```

    }

    if (pkt_debug)
    {
5       printf (" vt_field=%s, vt_table=%s, vt_min=%d, vt_max=%d\n",
        xtab_info[i].vt_field_name, xtab_info[i].vt_table_name,
        xtab_info[i].vt_min, xtab_info[i].vt_max);

        printf (" hz_field=%s, hz_table=%s, hz_min=%d, hz_max=%d\n",
        xtab_info[i].hz_field_name, xtab_info[i].hz_table_name,
        xtab_info[i].hz_min, xtab_info[i].hz_max);

        printf (" pg_field=%s, pg_table=%s, pg_min=%d, pg_max=%d\n",
10      xtab_info[i].pg_field_name, xtab_info[i].pg_table_name,
        xtab_info[i].pg_min, xtab_info[i].pg_max);
    }

    /* svp - check on this, not being used */
    dist_start_pos = pams_message;
    xtab_info_idx = 0;

    break;

15  case PKT_SEGMENTATION:
    /*
     * Segmentation Query!
     */

    num_tables = 1;
    if (pkt_simdbg)
    {
20      SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 2);
        printf ("SEPERATOR\n");
    }
    else
    {
        SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 2);
        if (pkt_debug)
            printf ("SEPERATOR\n");
    }

    num_of_dist = atoi(pams_message);

25  if (pkt_debug)
        printf ("number of distributions is %d\n", num_of_dist);

    pams_message = strchr(pams_message, '@') + 1;

    for (i = 0; i < num_of_dist; i++)
    {
        for (j=0; j < num_tables; j++)
        {
30          GET_FIELD_NAME();
        }
    }

    if (pkt_debug)
    {
        printf (" field=%s\n", dist_info[0].field_name);
    }

35  /* check to see if segmentation should be reset or not */
    segmentation_char = pams_message[0];
    segmentation_response = segmentation_char;

```

```

        if ((segmentation_response == 'Y') || (segmentation_response == 'y'))
            reset_segmentation = 1;
        else
            reset_segmentation = 0;

5         pams_message = strchr(pams_message, '@') + 1;

        /* svp - check on this, not being used. */
        dist_start_pos = pams_message;
        dist_info_idx = 0;

        break;

    } /* end switch (PACKET_TYPE) */

10    } /* end if (packet_processing) */

    if (DMQ_CONNECTED)
    {
        sprintf(temp_string85, "%s\n", ntwk_hdr->label);
        center_string(temp_string85, 70, 85); /* 70 char page width, 85 char input string */
        msg_buf_xfer(temp_string85);
    }
    else
        printf("%s\n", ntwk_hdr->label);

15    } /* end if ('msg_outstanding' and 'msg_not_outstanding') */
    else
    {
        /* check to see if pams_message is CIS_EOI else put CIS_EOI to that message */
        if (msg_outstanding)
            pams_put_msg_type = MSG_OUTSTANDING_TYPE;
        else
        {
20            msg_buf_xfer("No outstanding messages found\n");
            pams_put_msg_type = MSG_NO_OUTSTANDING;
        }
    }

    } /* end if (read_ntwk_hdr) */

    /*
    - Check for end of information
    */

25    if (*pams_message == CIS_EOI)
    {
        if (pkt_debug)
            printf("CIS_EOI\n");

        read_ntwk_hdr = TRUE;
        reset_parse_tree();

        if (DMQ_CONNECTED)
30        {
            dmq_put_msg();
            free_dmq_buffers();
            set_dmq_buffer();
            memset(msg_file_buffer, 0, FILE_BUFFER_LENGTH);
            init_dmq_buffers();

            /* reset the values */
            pams_put_msg_type = MSG_SINGLE_TYPE;
            msg_outstanding = 0;
            msg_not_outstanding = 0;

35    }

```

```

        initialize_variables();

        /* see if customer to purchase now mapped. if so unmap */
        /* note: not looking at delete_pur_info flag, assuming that flag
        cus_pur_map is set only if delete_pur_info */
5      if ( cus_pur_map )
      {
          status = UnmapCloseFile( cus_pur_chnl, &cus_pur_addr );
          if (is_error(status))
              error_handler( UNMAP_CLOSE_ERR, ERROR, status, "fdc_prototype, customer to purchase
              map count");
          cus_pur_map = 0;
      }

10     /* see if customer to product now mapped. if so unmap */
     if ( cus_pro_map )
     {
         status = UnmapCloseFile( cus_pro_chnl, &cus_pro_addr );
         if (is_error(status))
             error_handler( UNMAP_CLOSE_ERR, ERROR, status, "fdc_prototype, customer to product
             map count");
         cus_pro_map = 0;
     }

15     /* see if purchase to product now mapped. if so unmap */
     if ( pur_pro_map )
     {
         status = UnmapCloseFile( pur_pro_chnl, &pur_pro_addr );
         if (is_error(status))
             error_handler( UNMAP_CLOSE_ERR, ERROR, status, "fdc_prototype, purchase to product
             map count");
         pur_pro_map = 0;
     }

20     /*
     -- Continue ... This will wait for a DMO reconnection or
     -- get another line from stdio
     */

    continue; /* jump to end of while (TRUE) */

    /* end of CIS_EOI */

25     /*
     -- Process the query packet(s) in the parser'
     */

    PROCESS_QUERY_PACKETS
        query_num++;

        /* For multiple queries on a segmentation, do a reset only for the first query */
        if ((query_num > 1) && (reset_segmentation))
            reset_segmentation = 0;

30     if (packet_processing)
    {
        if (pkt_simdbg)
        {
            SEPERATOR_CHECK (pams_message++, DEBUG_SEPERATOR, 3);
            printf ("SEPERATOR\n");
        }
        else
        {
35         SEPERATOR_CHECK (pams_message++, PACKET_SEPERATOR, 3);
            if (pkt_debug)
                printf ("SEPERATOR\n");
        }
    }

```

```

/*
 * Delete the keycode and label from last time
 * If this is not a quick query, get the keycode and
 * label from the input stream.
 */
5
if (strlen(query_keycode) > 0)
{
    free(query_keycode);
    query_keycode = NullString;
}

if (strlen(query_label) > 0)
10
{
    free(query_label);
    query_label = NullString;
}

if (nwsk_pkt_type != PKT_QUICK_QUERY)
{
    len = (int) strchr(pams_message, '@');
    if (len != 0)
    {
15
        len = (int) pams_message; /* length of keycode */
        query_keycode = malloc(len + 1); /* room for the '\0' */
        CHECK_ALLOCATION(query_keycode, "query_keycode: Routine main()");
        memcpy(query_keycode, pams_message, len); /* copy keycode */
        query_keycode[len] = '\0'; /* insert the '\0' */

        pams_message += (len + 1); /* skip past the '@' */

        len = (int) strchr(pams_message, '@');
20
        if (len != 0)
        {
            len = (int) pams_message; /* length of keycode */
            query_label = malloc(len + 1); /* room for the '\0' */
            CHECK_ALLOCATION(query_label, "query_label: Routine main()");
            memcpy(query_label, pams_message, len); /* copy label */
            query_label[len] = '\0'; /* insert the '\0' */

            pams_message += (len + 1); /* skip past the '@' */
        }
    }
25
}

/*
 * YYPARSE is currently used exclusively for processing
 * query grammar and grammar related commands
 */

if (pkt_debug)
30
    printf("parsing query (keycode=%s label=%s)\n", query_keycode, query_label);

status = yyparse();

switch (status)
{
    case EXIT
        if (!DMQ_CONNECTED)
        {
35
            print_parse_history();
            exit_normal();
        }
    }
}

```



```

    /* Reset DMQ */

    dmq_put_msg();
    free_dmq_buffers();
    set_dmq_buffer();
5    memset(msg_file_buffer, 0, FILE_BUFFER_LENGTH);
    init_dmq_buffers();
    reset_parse_tree();

    /* reset the values */
    pams_put_msg_type = MSG_SINGLE_TYPE;
    msg_outstanding = 0;
    msg_not_outstanding = 0;
    query_num = 0;

    initialize_variables();
10    break;

    case QUERY :
    case UNIVERSE :
        resolve_query_tree(current_expr);
        add_parse_history();
        reset_parse_tree();
        row_count++;
        break;

15    case DISTRIBUTION :
    case DISPLAY :
        reset_parse_tree();
        row_count++;
        break;

    case UEND :
        if (!print_universe_history())
            printf("Error while closing the universe\n");
20    else
        reset_parse_tree();
        break;

    case SAMPLE_END :
        end_sampling();
        break;

    default:
25    destroy_query_tree();
    reset_parse_tree();
    break;

} /* end switch (status) */

    switch (nlwk_pkt_type)
    {
    case PKT_DISPLAY :

30    /* Save the current location of the pams_message.

    */

    save_pams_msg = pams_message;

    for (i = 0; i < num_of_dist; i++)
    {
        /*
        Build the distribution query.
        */
35    sprintf(customized_query, "DP %d:", i);

```

```

        /*
        * Point the pams_message to the customized_query
        */
        pams_message = customized_query;

5
        /*
        * Call the parser and process the distribution
        */

        if (pkt_debug)
            printf ("calling the parser with distribution # %d\n", i);

        yyparse();

10
    } /* end for (i < num_of_dist) */

    /*
    * Reset the pams_message
    */

    pams_message = save_pams_msg;
    break;

case PKT_WRITE :

15
    /*
    * Save the current location of the pams_message
    */

    save_pams_msg = pams_message;

    /*
    * Build the distribution query
    */

20
    sprintf (customized_query, "WR %d.", num_of_dist);

    /*
    * Point the pams_message to the customized_query
    */

    pams_message = customized_query;

    /*
    * Call the parser and process the distribution
    */

25
    if (pkt_debug)
        printf ("calling the parser with distribution # %d\n", i);

    yyparse();

    /*
    * Reset the pams_message
    */

30
    pams_message = save_pams_msg;
    break;

case PKT_DISTRIBUTION

    /*
    * Save the current location of the pams_message
    */

35
    save_pams_msg = pams_message;

    for (i = 0, i < num_of_dist, i++)
    {
        /*
        * Build the distribution query
        */
    }

```

```

    sprintf (customized_query, "display distribution %d:", i);

    /*
    - Point the pams_message to the customized_query.
    */
5   pams_message = customized_query;

    /*
    - Call the parser and process the distribution!
    */

    if (pkt_debug)
10      printf ("calling the parser with distribution # %d\n", i);

        yyparse();

        /* end for (i < num_of_dist) */

        /*
        - Reset the pams_message!
        */
15      pams_message = save_pams_msg;
        break;

case PKT_COND_DIST:

    /*
    - Save the current location of the pams_message
    */

    save_pams_msg = pams_message;
20    for (i = 0; i < num_of_dist; i++)
    {
        /*
        - Build the distribution query
        */

        sprintf (customized_query, "CD %d:", i);

        /*
        - Point the pams_message to the customized_query
        */
25      pams_message = customized_query;

        /*
        - Call the parser and process the distribution!
        */

        if (pkt_debug)
30          printf ("calling the parser with distribution # %d\n", i);

        yyparse();

        /* end for (i < num_of_dist) */

        /*
        - Reset the pams_message!
        */
35      pams_message = save_pams_msg;
        break;

```

```

case PKT_2WAY_XTABS:
    /*
     * Save the current location of the pams_message
     */

    5   save_pams_msg = pams_message.

    for (i = 0; i < num_of_dist; i++)
    {
        /*
         * Build the distribution query
         */

        sprintf (customized_query, "X2 %d.", i);

    10   /*
         * Point the pams_message to the customized_query
         */

        pams_message = customized_query;

        /*
         * Call the parser and process the distribution!
         */

    15   if (pkt_debug)
        printf ("calling the parser with distribution # %d\n", i);

        yyparse();

    } /* end for (i < num_of_dist) */

    /*
     * Reset the pams_message!
     */

    20   pams_message = save_pams_msg.
        break.

case PKT_COND_2WAY

    /*
     * Save the current location of the pams_message
     */

    25   save_pams_msg = pams_message.

    for (i = 0; i < num_of_dist; i++)
    {
        /*
         * Build the distribution query
         */

        30   sprintf (customized_query, "C2 %d.", i);

        /*
         * Point the pams_message to the customized_query
         */

        pams_message = customized_query;

        /*
         * Call the parser and process the distribution!
         */

    35   if (pkt_debug)
        printf ("calling the parser with distribution # %d\n", i);

        yyparse();

```

```

    } /* end for (i < num_of_dist) */

    /*
    -- Reset the pams_message!
    */

5      pams_message = save_pams_msg;
      break;

case PKT_3WAY_XTABS :
    /*
    -- Save the current location of the pams_message.
    */

10     save_pams_msg = pams_message;

    for (i = 0; i < num_of_dist; i++)
    {
        /*
        -- Build the distribution query.
        */

15         sprintf (customized_query, "X3 %d:", i);

        /*
        -- Point the pams_message to the customized_query
        */

        pams_message = customized_query;

        /*
        -- Call the parser and process the distribution!
        */

20         if (pkt_debug)
            printf ("calling the parser with distribution #%d\n", i);

        yyparse();

    } /* end for (i < num_of_dist) */

    /*
    -- Reset the pams_message!
    */

25     pams_message = save_pams_msg;
    break;

case PKT_COND_3WAY

30     /*
    -- Save the current location of the pams_message
    */

    save_pams_msg = pams_message;

    for (i = 0; i < num_of_dist; i++)
    {
        /*
        -- Build the distribution query
        */

35         sprintf (customized_query, "C3 %d:", i);

        /*
        -- Point the pams_message to the customized_query.
        */

```

```

        pams_message = customized_query;

        /*
         * Call the parser and process the distribution!
         */

5         if (pkt_debug)
            printf ("calling the parser with distribution # %d\n", i);

        yyparse();

    } /* end for (i < num_of_dist) */

    /*
     * Reset the pams_message!
     */

10    pams_message = save_pams_msg;
    break;

case PKT_RFM :

    /*
     * Save the current location of the pams_message
     */

15    save_pams_msg = pams_message;

    for (i = 0; i < num_of_dist; i++)
    {
        /*
         * Build the distribution query.
         */

        sprintf (customized_query, "XRFM %d;", i);

20        /*
         * Point the pams_message to the customized_query
         */

        pams_message = customized_query;

        /*
         * Call the parser and process the distribution!
         */

25        if (pkt_debug)
            printf ("calling the parser with distribution # %d\n", i);

        yyparse();

    } /* end for (i < num_of_dist) */

    /*
     * Reset the pams_message!
     */

30    pams_message = save_pams_msg;
    break;

case PKT_SEGMENTATION :

    /*
     * Save the current location of the pams_message
     */

35    save_pams_msg = pams_message;

    for (i = 0; i < num_of_dist; i++)
    {

```

```

/*
 * Build the segmentation query
 */

5      sprintf (customized_query, "SG %d", i);

/*
 * Point the pams_message to the customized_query
 */

      pams_message = customized_query;

/*
 * Call the parser and process the distribution
 */

10     if (pkt_debug)
        printf ("calling the parser with segmentation # %d\n", i);

        yyparse();

        /* end for (i < num_of_dist) */

/*
 * Reset the pams_message
 */

15     pams_message = save_pams_msg;
        break;

    /* end of switch statement */

error_exception

    /* end while (TRUE) */

    /* end of main() */

20

void null_terminate (char *string, int length)
{
25     int i;

    for (i = (length - 1); i >= 0; i--)
        if (string[i] != '\0')
            break;

    if (i < (length - 1))
        string[i + 1] = '\0';
    else
30         string[i] = '\0';

    /* end of null_terminate() */

void load_config_file (config_file)
char *config_file;
{
    FILE *f; /* token types file */
    char line[255];
35     char token_name[MAX_TOKEN_NAME_LEN], token_value[MAX_TOKEN_VALUE_LEN];
    char *ptr;
    int i, j;
    int number_of_tokens;

```



```

/*
 * Create a hash table for the tokens from the config file.
 */

ht = hash_init(32);

5
if ((tf = fopen(config_file, "r")) == NULL)
{
    error_handler(FILE_NOT_OPEN, ERROR, NO_STATUS, config_file);
}

sptr = fgets(line, sizeof(line), tf);

while (sptr != NULL)
{
10
    sscanf(line, "%25s%s", token_name, token_value);

    /*
     * Strip trailing spaces off of the token name and value.
     */

    if ((sptr = strchr(token_name, '\0')) != NULL)
        *sptr = '\0';

15
    if ((sptr = strchr(token_value, '\0')) != NULL)
        *sptr = '\0';

    /*
     * Insert the token value into the hash table and allocate enough
     * space for the token name (including a '\0'). Copy the token
     * name into the memory allocated by the hash insert function to
     * associate it with the token value
     */

20
    sptr = (char *) hash_insert(ht, token_value, strlen(token_name) + 1);
    strcpy(sptr, token_name);

    /*
     * Get the next line from the config file
     */

    sptr = fgets(line, sizeof(line), tf);

25
} /* end while (sptr != NULL); */

/* print hash table (ht). */

fclose(tf);
}

free_amq_buffers()
{
30
    int i;

    /* initialize the buffers */
    for (i = 0; i <= msg_offset; i++)
    {
        /* should free all buffers but 1 */
        if (msg_put_buffer[msg_offset] != NULL)
        {
35
            free(msg_put_buffer[msg_offset]);
            msg_put_buffer[msg_offset] = NULL;
        }
    }
}

```

```

5      set_dmz_buffer()
      {
          msg_offset = 0;
          msg_put_buffer(msg_offset) = malloc(BUFFER_LENGTH);
          CHECK_ALLOCATION(msg_put_buffer(msg_offset), "msg_put_buffer: Routine set_dmz_buffer()");
          memset(msg_put_buffer(msg_offset), 0, BUFFER_LENGTH);
          memset(msg_summary_buffer, 0, BUFFER_LENGTH);
      }

10

      int_dmz_buffers()
      {
          int i;

          /* initialize buffers */
          for (i = 1; i < MAX_BUFFERS; i++)
              msg_put_buffer(i) = NULL;
      }

15

      void initialize_variables()
      {
          /* initialize the super_master_bitmap */
          if (super_master_bitmap)
          {
20              free_bitmap(super_master_bitmap);
              super_master_bitmap = NULL;
          }

          /* initialize the other variables */
          row_count = 1;
          read_nwz_hdr = TRUE;
          initial_occurrence = TRUE;

25      }

      int upper_strcmp(s, t)
      char *s;
      char *t;
      {
30          if (s == NULL) return(-1);
          if (t == NULL) return(-1);

          while(toupper(*s) == toupper(*t))
          {
              if (*s == '\0') return(0);
              s++;
              t++;
          }
          return (*s - *t);
35      }

```

```

5  /* Common error handler routine. All errors due to normal and abnormal conditions
   * must come through here. The routine will send error information to both system
   * and the PC client when needed. A clean up is also done to prepare us for a reset
   * or exit. Only fatal errors will cause an exit call from this routine. Non fatal
   * errors should have a method of bringing us back to the While (TRUE) condition in
   * FDC_PROTOTYPE.C. Try to have all calls have some kind of continue method after
   * the error_handler()
   */

void error_handler (int message_id,          /* As defined in FDC_ERROR_NUMBERS */
                   int message_type,         /* ERROR or WARNING_ONLY */
                   int additional_info_int,  /* May not always be present */
                   char *additional_info_string) /* May not always be present */

10  /* BEGIN error handler */

    int how_handle_error=0;
    char *error_msg_buffer;

    if (message_type == ERROR)
    {
        /* Clean up before we exit or continue. Not all the items
        /* have to be done for the exit condition. But since the
        /* exit will kill the process anyway it just doesn't matter. */

15        read_nwk_hdr = TRUE;
        reset_parse_tree();

        if (DMQ_CONNECTED)
        {
            /* free the existing msg information, since an error has occurred */

            free_dmq_buffers();
            set_dmq_buffer();
            memset(msg_file_buffer, 0, FILE_BUFFER_LENGTH);
            init_dmq_buffers();

            /* reset the values */
            pams_put_msg_type = MSG_SINGLE_TYPE;
            msg_outstanding = 0;
            msg_not_outstanding = 0;
        }

20        initialize_variables();

        /* endif ERROR */ /* Note - for now nothing is done if message_type is WARNING_ONLY CM */

        /* call to error_msg_routine to print out
        /* error messages and determine severity of error */

        how_handle_error = error_msg_routine (message_id, message_type, additional_info_int, additional_info_string);

25        if (how_handle_error == FATAL_EXIT)
            exit(1); /* kill the fdc_prototype c process - not normal exit */
        else if (how_handle_error == EXIT_NORMAL)
            exit(0); /* normal exit with no DMQ CONNECTED */
        else if (how_handle_error == NON_FATAL_RETRY)
            return; /* do nothing return to call !! original call expects return */
        else
            exit(1); /* for some reason did not return a valid default */

30        /* END error handler */

35

```

```
exit_normal()
{
    error_handler(NORMAL_EXIT, ERROR, NO_STATUS, NO_STRING); /* this routine already does all our cleanup work */
#ifdef DBG
5   fclose("debug_file");
#endif
}

10   long check_ret(sts$value)
    {
        if (sts$value & STSSM_SUCCESS)
            return (1);

        printf ("PAMS ERROR RETURNED 0x%x\n", sts$value);
        return(0);
    }

15

20

25

30

35
```

5

10

Appendix B for U.S. Patent Application

15

DATABASE LINK SYSTEM
BACKGROUND OF THE INVENTION

Filed October 22, 1993

Appendix B: Routines to Handle Purchase
and Product Query Manipulation

20

© 1993 FDC, Inc.
All Rights Reserved

25

30

35

```

/*
**
** MODULE: PUR_PRD_QUERY.C
**
** MODULE DESCRIPTION.
**
5  ** Set of routines to handle purchase and product query manipulation.
**
** AUTHORS:
**
**         Kelly Westman           Digital Equipment Corporation
**         Sushil Pillai          Digital Equipment Corporation
**         Mike Emerson/Dan Thomas
**
** CREATION DATE: 8-February-1993
**
10  ** DESIGN ISSUES.
**
**         Performs a linear search through a dataset. Does not do any sorting
**         or presupposes a sorted order to data.
**         Search routines depend on datatype of field being searched - avoided
**         function calls in middle of loop for speed.
**
** PORTABILITY ISSUES:
**
15  **         Uses VMS system calls for create, mapping and unmapping global sections
**         Uses RMS attributes blocks for file to open
**
** MODIFICATION HISTORY:
**
**         8-February-1993 - Original
**         17-May-1993   - Chuck Matmskog
**         - Replaced fail_and_exit calls with error_handler calls
**         22-Sep-1993   - SVP
**         - Included OR condition for first, second, third, any and domain purchase/product queries
20  **         23-Sep-1993   - SVP
**         - Rework on average and total purchase/product queries
**         30-Sep-1993   - Charles Malmkog
**         - Update error_handler() calls to include ERROR message type
**
/*
**
25  ** INCLUDE FILES
**
**
** #include <stdio.h>
** #include <stdlib.h>
** #include <ssdef.h>
** #include <rms.h>
** #include <tab.h>
** #include <secdef.h>
** #include <psidef.h>
30  ** #include <odescrip.h>
** #include <unixio>
** #include <file>
** #include <math>
**
** #include "tdc_prototype"
** #include "tdc_macro_defn"
** #include "tdc_error_numbers.h"
**
35  /* external declarations */
/* structures for fixed strings */

extern struct fixed_string_type fixed_field[NUM_FIXED_STRINGS];
extern int max_number_of_bits.

```

```

extern int max_purchase_bits;
extern int max_product_bits;

blend = setbit + BITMAP_INTEGER_SIZE;

5  /* DMQ specific */
extern int DMQ_CONNECTED;

/* structures for bitmaps */
extern struct bitmap *subsidiary_bitmap(SUB_FILE_MAX);
extern struct bitmap *pur01_bitmap;
extern unsigned short *pur01_map;
extern unsigned int pur01_map_count;

10 extern struct bitmap *prd01_bitmap;
extern unsigned short *prd01_map;
extern unsigned int prd01_map_count;
extern struct bitmap *master_purchase_bitmap;
extern struct bitmap *master_product_bitmap;

/* declared in search_section, signifies if a purchase query or product query is in use */
extern int purchase_query;
extern int product_query;

15 /* for debug purposes */
extern FILE *debug_file;

/*
 * Function prototypes
 */

void dom_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
20 int any_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
int first_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
int second_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
int third_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
25 int multiple_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
int any_2_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
any_3_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
int last_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
int ntl_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
30 int last2_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );
int last5_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap,
enum boolean_operator oper, struct bitmap *results_bitmap );

int avg_pur_prd_large (unsigned int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );
35 int total_pur_prd_large (unsigned int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *pur_map, struct bitmap *pur_prd_bitmap );

```



```

int (*pur_prd_select[12])( struct bitmap *p_bitmap, struct bitmap *pur_prd_bitmap,
                           enum boolean_operator oper, struct bitmap *results_bitmap );

5  int (*pur_prd_large_special[2])(unsigned int *data, unsigned int value, unsigned int high_value, struct list_types *list,
                                   enum field_operator operator, int num_items, struct bitmap *results_map,
                                   unsigned short *map, struct bitmap *pur_prd_bitmap );

int avg_pur_prd_medium (unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
                        enum field_operator operator, int num_items, struct bitmap *results_map,
                        unsigned short *map, struct bitmap *pur_prd_bitmap );
int total_pur_prd_medium (unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
                          enum field_operator operator, int num_items, struct bitmap *results_map,
                          unsigned short *map, struct bitmap *pur_prd_bitmap );
10

int (*pur_prd_medium_special[2])(unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
                                  enum field_operator operator, int num_items, struct bitmap *results_map,
                                  unsigned short *map, struct bitmap *pur_prd_bitmap );

int avg_pur_prd_date (unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
                      enum field_operator operator, int num_items, struct bitmap *results_map,
                      unsigned short *map, struct bitmap *pur_prd_bitmap );
15
int total_pur_prd_date (unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
                        enum field_operator operator, int num_items, struct bitmap *results_map,
                        unsigned short *map, struct bitmap *pur_prd_bitmap );

int (*pur_prd_date_special[2])(unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
                                enum field_operator operator, int num_items, struct bitmap *results_map,
                                unsigned short *map, struct bitmap *pur_prd_bitmap );
20

int avg_pur_prd_short (unsigned char *data, unsigned int value, unsigned int high_value, struct list_types *list,
                       enum field_operator operator, int num_items, struct bitmap *results_map,
                       unsigned short *map, struct bitmap *pur_prd_bitmap );
int total_pur_prd_short (unsigned char *data, unsigned int value, unsigned int high_value, struct list_types *list,
                          enum field_operator operator, int num_items, struct bitmap *results_map,
                          unsigned short *map, struct bitmap *pur_prd_bitmap );

25
int (*pur_prd_short_special[2])(unsigned char *data, unsigned int value, unsigned int high_value, struct list_types *list,
                                 enum field_operator operator, int num_items, struct bitmap *results_map,
                                 unsigned short *map, struct bitmap *pur_prd_bitmap );

int avg_pur_prd_char (char *data, unsigned int value, unsigned int high_value, struct list_types *list,
                      enum field_operator operator, int num_items, struct bitmap *results_map,
                      unsigned short *map, struct bitmap *pur_prd_bitmap );

30
int total_pur_prd_char (char *data, unsigned int value, unsigned int high_value, struct list_types *list,
                        enum field_operator operator, int num_items, struct bitmap *results_map,
                        unsigned short *map, struct bitmap *pur_prd_bitmap );

int (*pur_prd_char_special[2])(char *data, unsigned int value, unsigned int high_value, struct list_types *list,
                                enum field_operator operator, int num_items, struct bitmap *results_map,
                                unsigned short *map, struct bitmap *pur_prd_bitmap );
35

```

```

int avg_pur_prd_string (char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );
int total_pur_prd_string (char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int (*pur_prd_string_special[2])(char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int avg_pur_prd_mixed_string (char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );
int total_pur_prd_mixed_string (char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int (*pur_prd_mixed_string_special[2])(char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int avg_pur_prd_fstring (unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );
int total_pur_prd_fstring (unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int (*pur_prd_fstring_special[2])(unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int avg_pur_prd_mixed_fstring (unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );
int total_pur_prd_mixed_fstring (unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int (*pur_prd_mixed_fstring_special[2])(unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int avg_pur_prd_bit( int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );
int total_pur_prd_bit( int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

int (*pur_prd_bit_special[2])(int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap );

```

```

int pur_prd_query_special( struct dataset *data, enum data_type field_type, int field_size
                           int field_offset, enum field_operator operator, union search_item search_value
                           enum search_value_item search_value_type, enum pur_prd_query_type query_type
                           struct bitmap *results_bitmap);

5 void pur_prd_query( enum pur_prd_query_type query_type, struct bitmap *results_bitmap);
{
    pur00 = purchase_domain
    pur01 = any_pur_prd
    pur02 = first_pur_prd
    pur03 = second_pur_prd
    pur04 = third_pur_prd
    pur05 = multiple_pur_prd
10 pur06 = two_pur_prd
        = any_2_pur_prd
    pur07 = three_pur_prd
        = any_3_pur_prd
    pur08 = last_pur_prd
    pur09 = ntl_pur_prd
    pur10 = last2_pur_prd
    pur11 = last5_pur_prd
    pur12 = avg_pur_prd
15 pur13 = total_pur_prd
}

init_pur_prd_table ( )
{
    pur_prd_select[0] = dom_pur_prd_select;
    pur_prd_select[1] = any_pur_prd_select;
    pur_prd_select[2] = first_pur_prd_select;
20 pur_prd_select[3] = second_pur_prd_select;
    pur_prd_select[4] = third_pur_prd_select;
    pur_prd_select[5] = multiple_pur_prd_select;
    pur_prd_select[6] = any_2_pur_prd_select;
    pur_prd_select[7] = any_3_pur_prd_select;
    pur_prd_select[8] = last_pur_prd_select;
    pur_prd_select[9] = ntl_pur_prd_select;
    pur_prd_select[10] = last2_pur_prd_select;
    pur_prd_select[11] = last5_pur_prd_select;

25 pur_prd_large_special[0] = avg_pur_prd_large;
    pur_prd_large_special[1] = total_pur_prd_large;

    pur_prd_medium_special[0] = avg_pur_prd_medium;
    pur_prd_medium_special[1] = total_pur_prd_medium;

    pur_prd_date_special[0] = avg_pur_prd_date;
    pur_prd_date_special[1] = total_pur_prd_date;

30 pur_prd_short_special[0] = avg_pur_prd_short;
    pur_prd_short_special[1] = total_pur_prd_short;

    pur_prd_char_special[0] = avg_pur_prd_char;
    pur_prd_char_special[1] = total_pur_prd_char;

    pur_prd_string_special[0] = avg_pur_prd_string;
    pur_prd_string_special[1] = total_pur_prd_string;

35 pur_prd_mixed_string_special[0] = avg_pur_prd_mixed_string;
    pur_prd_mixed_string_special[1] = total_pur_prd_mixed_string;

    pur_prd_fstring_special[0] = avg_pur_prd_fstring;
    pur_prd_fstring_special[1] = total_pur_prd_fstring;
}

```

```

    pur_prd_mixed_fstring_special[0] = avg_pur_prd_mixed_fstring;
    pur_prd_mixed_fstring_special[1] = total_pur_prd_mixed_fstring;

    pur_prd_bit_special[0] = avg_pur_prd_bit;
5    pur_prd_bit_special[1] = total_pur_prd_bit;
}

void dom_pur_prd_select (
    struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
10 {
    combine_bitmaps(pur_prd_bitmap, oper, next_pur_prd_bitmap);
    if (copy_bitmaps(results_bitmap, pur_prd_bitmap) == NULL)
        error_handler (BITMAP_NOT_COPY, ERROR, NO_STATUS, "pur_prd_bitmap to results_bitmap in
        <dom_pur_prd_select
    )

/*
-- CHECK_PUR_PRD_BITS
-- Check_pur_prd_bits performs an bitwise logical operations on two bits and returns either a 1 or a 0.
15 -- FORMAT
--
-- int check_pur_prd_bits()
--
-- ARGUMENTS
--
-- unsigned int pur_prd_bit          - bit pointed to by the bitmap1
-- enum boolean_operator oper        - logical operator
20 -- unsigned int pur_prd_const_bit  - bit pointed to by the constant bitmap, these bits are all 1
--
-- RETURN VALUES
--
-- 1 or a 0, is SUCCESS or a FAILURE
--
-- SIDE EFFECTS
--
-- If an operation fails, will fail and exit the program
*/
25 int check_pur_prd_bits (unsigned int pur_prd_bit, enum boolean_operator oper, unsigned int pur_prd_const_bit)
{
    switch (oper)
    {
        case or_when :
        case or :
            if (pur_prd_bit | pur_prd_const_bit)
                return(1);
            else
30             return(0);
            break;
        case and_when :
        case and :
            if (pur_prd_bit & pur_prd_const_bit)
                return(1);
            else
                return(0);
            break;
        default :
35             error_handler (INVALID_SWITCH_VALUE, ERROR, oper, "oper in check_pur_prd_bits");
            break;
    }
}

```

```

int any_pur_prd_select( struct bitmap *pur_prd_bitmap, struct bitmap *next_pur_prd_bitmap
                        enum boolean_operator oper, struct bitmap *results_bitmap )
{
    PUR_PRD_SEL_VARS;
5    PUR_PRD_SEL_PRELIM;

    results = results_bitmap->start;

    /*
    pur_prd_bitmap is the pur_prd bitmap (size 5M).
    We need to "and" this bitmap with the next_pur_prd_bitmap to give a any_pur_prd bitmap
    ie: combine_bitmaps(pur_prd_bitmap, and, next_pur_prd_bitmap)

    we can then reduce it to the customer level
    Note: next_pur_prd_bitmap is a bitmap with all bits set (for any pur_prd)
    We will have to create pur02_bitmap with only the first bit set (for first pur_prd), etc
    */
    pur_prd_map = pur_prd_bitmap->start;
    pur_defn_map = next_pur_prd_bitmap->start;

    for(count=0; map < map_end; map++)
    {
15         if ( (j = *map) >= 1 )
            {
                for ( i=0; i < j; i++)
                {
                    /* verify item defined in next_pur_prd_bitmap for this customer */
                    /*
                    pur_prd_bits is pointing to a constant bitmap with all bits set.
                    pur_defn_map is pointing to bitmap1 (which is being used as the reference bitmap).
                    pur_prd_map is pointing to bitmap2.
                    */
20                 if (check_pur_prd_bits ((*pur_defn_map & *pur_prd_bits), oper, (*pur_prd_map
                    & *pur_prd_bits)))
                    {
                        count++;
                        *results |= *pur_prd_bits;
                    }
                    NEXT_PUR_PRD_RESULT_BIT1;
                }
            }
    }
    return(count);
25 }

int first_pur_prd_select(
                        struct bitmap *pur_prd_bitmap
                        struct bitmap *next_pur_prd_bitmap,
                        enum boolean_operator oper,
                        struct bitmap *results_bitmap
                        )
{
30    PUR_PRD_SEL_VARS;
    PUR_PRD_SEL_PRELIM;

    results = results_bitmap->start;

    pur_prd_map = pur_prd_bitmap->start;
    pur_defn_map = next_pur_prd_bitmap->start;

35    for(count=0; map < map_end; map++)
    {

```

```

    if ( (j = *map) >= 1 )
    {
        for ( i=0; i < j; i++)
        {
            if (check_pur_prd_bits ((*pur_defn_map & *pur_prd_bits), oper, (*pur_prd_map & *pur_prd_bits)))
            {
                count++;
                *results |= *pur_prd_bits;
                break;
            }
            NEXT_PUR_PRD_RESULT_BIT1;
        }
        for ( ; i < j; i++)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
}
10 return(count);
}

int second_pur_prd_select(
    struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
)
{
15 PUR_PRD_SEL_VARS;
    PUR_PRD_SEL_PRELIM;

    results = results_bitmap->start;

    pur_prd_map = pur_prd_bitmap->start;
    pur_defn_map = next_pur_prd_bitmap->start;
    for(count=0, map < map_end, map++)
    {
20         if ( (j = *map) >= 2 )
        {
            for ( current=0, i=0, i<j; i++)
            {
                /* looks at 2nd item defined in next_pur_prd_bitmap for each customer */
                if (*pur_defn_map & *pur_prd_bits)
                {
                    current1++;
                }
                if (*pur_prd_bits & *pur_prd_map)
                {
                    current2++;
25
                    switch(oper)
                    {
                        case and_when
                        case and
                    }

                    if ( current1 == 2 )
                    {
                        /* now see if 2nd item met the search criteria */
                        if (*pur_prd_bits & *pur_prd_map)
                        {
30                             count++;
                             *results |= *pur_prd_bits;
                        }
                        /* now that we've looked at 2nd defined pur_prd item this customer stop looking */
                        break;
                    }
                }
            }
        }
    }
}
35

```

```

    }
    break;
case or_when :
case or :
5      if (( current == 2 ) || (current2 == 2))
      {
          count++;
          *results |= *pur_prd_bits;
          /* now that we've looked at 2nd defined pur_prd item this customer stop looking */
          break;
      }
      break;
default :
10      error_handler (INVALID_SWITCH_VALUE, ERROR, oper, "oper in check_pur_prd_bits");
      break;
    }
    NEXT_PUR_PRD_RESULT_BIT1;
    for ( ; i < j; i++)
    {
        NEXT_PUR_PRD_RESULT_BIT1;
    }
15    }
    else
    {
        /* not enuf records. skip past 1 bit if we need to */
        if (j == 1)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
    }
20    return(count);
}

int third_pur_prd_select(
    struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
)
{
25    PUR_PRD_SEL_VARS;
    PUR_PRD_SEL_PRELIM;
    results = results_bitmap->start;
    pur_prd_map = pur_prd_bitmap->start;
    pur_defn_map = next_pur_prd_bitmap->start;
    for(count=0; map < map_end; map++)
30    {
        if (j = *map) >= 3)
        {
            for ( current=0, i=0; i < j; i++)
            {
                /* looks at 3rd item defined in next_pur_prd_bitmap for each customer */
                if (*pur_defn_map & *pur_prd_bits)
                    current++;
                if (*pur_prd_bits & *pur_prd_map)
                    current2++;
35
            }
        }
    }
}

```



```

switch(oper)
{
    case and_when :
        case and :
            5
                if( current == 3 )
                {
                    /* now see if 3rd item met the search criteria */
                    if ( *pur_prd_bits & *pur_prd_map )
                    {
                        count++;
                        *results |= *pur_prd_bits;
                    }
                    /* now that we've looked at 3rd defined pur_prd item this customer stop looking */
                    break;
                }
            10
                case or_when :
                case or :
                    if (( current == 3 ) || (current2 == 3))
                    {
                        count++;
                        *results |= *pur_prd_bits;
                        /* now that we've looked at 3rd defined pur_prd item this customer stop looking */
                        break;
                    }
            15
                default :
                    error_handler (INVALID_SWITCH_VALUE, ERROR, oper, "oper in check_pur_prd_bits");
                    break;
            }
            NEXT_PUR_PRD_RESULT_BIT1.
        }
        for ( i = 1; i++ )
        {
            NEXT_PUR_PRD_RESULT_BIT1.
        }
        else
        {
            /* not enuf records. skip past 0, 1, or 2 bits */
            for ( i = 0; i < j; i++ )
            25
                NEXT_PUR_PRD_RESULT_BIT1.
        }
    }
    return(count);
}

int multiple_pur_prd_select(
    30
    struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
    )
{
    unsigned int *sav_bit1, *sav_res1;
    PUR_PRD_SEL_VARS;
    PUR_PRD_SEL_PRELIM;
    35
    results = results_bitmap->start;

```

```

/* Going to mark all multiple purchase records, for each customer */
pur_prd_map = pur_prd_bitmap->start;
pur_defn_map = next_pur_prd_bitmap->start;
5 for(count=0; map < map_end; map++)
{
    if ( (j = *map) >= 2 )
    {
        for ( found=0, i=0; i<j; i++)
        {
            /* looks at only items defined in next_pur_prd_bitmap for each customer */
            if (*pur_defn_map & *pur_prd_bits)
            {
                /* now see item met the search criteria */
                if (*pur_prd_bits & *pur_prd_map)
                {
                    found++;
                    if ( found == 1 )
                    {
                        sav_res1 = results;
                        sav_bit1 = pur_prd_bits;
                    }
                    /* count it if at least the 2nd item */
                    if ( found > 1 )
                    {
                        count++;
                        *results |= *pur_prd_bits;
                        if ( found == 2 )
                        {
                            *sav_res1 |= *sav_bit1;
                            count++;
                        }
                    }
                }
            }
        }
        NEXT_PUR_PRD_RESULT_BIT1;
    }
    else
    {
        /* not enuf records. skip past 0, or 1 bits */
        for ( i=0; i<j; i++)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
}
return(count);
}

int any_2_pur_prd_select(
30
{
    struct bitmap *pur_prd_bitmap;
    struct bitmap *next_pur_prd_bitmap;
    enum boolean_operator oper;
    struct bitmap *results_bitmap;
    unsigned int *sav_bit1, *sav_res1;
    PUR_PRD_SEL_VARS;
35

```

```

PUR_PRD_SEL_PRELIM,

results = results_bitmap->start;

pur_prd_map = pur_prd_bitmap->start;
pur_defn_map = next_pur_prd_bitmap->start;
5 for(count=0; map < map_end; map++)
{
    if ( (j = *map) >= 2 )
    {
        for ( found=0, i=0; i<j; i++)
        {
            /* looks at only items defined in next_pur_prd_bitmap for each customer */
            if (*pur_defn_map & *pur_prd_bits)
            {
                /* now see item met the search criteria */
                if ( *pur_prd_bits & *pur_prd_map )
                {
                    found++;
                    /* count it if at least the 2nd item */
                    if ( found == 1 )
                    {
                        sav_res1 = results;
                        sav_bit1 = pur_prd_bits;
                    }
                    else
                    {
                        count++;
                        *results |= *pur_prd_bits;
                        if ( found == 2 )
                        {
                            count++;
                            *sav_res1 |= *sav_bit1;
                        }
                    }
                }
            }
        }
        NEXT_PUR_PRD_RESULT_BIT1;
    }
    else
    {
        /* not enuf records skip past 0, or 1 bits */
        for ( i=0; i<j; i++)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
}
return(count);
}

int any_3_pur_prd_select(
30 struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
)
{
    unsigned int *sav_bit1, *sav_bit2, *sav_res1, *sav_res2;

    PUR_PRD_SEL_VARS:

    PUR_PRD_SEL_PRELIM,
35 results = results_bitmap->start;

```

```

pur_prd_map = pur_prd_bitmap->start;
pur_defn_map = next_pur_prd_bitmap->start;
for(count=0; map < map_end; map++)
{
    5      if ( (j = *map) >= 3 )
    {
        for ( found=0, i=0; i<j; i++)
        {
            /* looks at only items defined in next_pur_prd_bitmap for each customer */
            if ( *pur_defn_map & *pur_prd_bits )
            {
                /* now see item met the search criteria */
                if ( *pur_prd_bits & *pur_prd_map )
                {
                    10      found++;
                    /* count it if at least the 3rd item */
                    if ( found == 1 )
                    {
                        sav_res1 = results;
                        sav_bit1 = pur_prd_bits;
                    }
                    else
                    {
                        15      if ( found == 2 )
                        {
                            sav_res2 = results;
                            sav_bit2 = pur_prd_bits;
                        }
                        else
                        {
                            count++;
                            *results |= *pur_prd_bits;
                            if ( found == 3 )
                            {
                                20      count++;
                                *sav_res1 |= *sav_bit1;
                                count++;
                                *sav_res2 |= *sav_bit2;
                            }
                        }
                    }
                }
            }
        }
        NEXT_PUR_PRD_RESULT_BIT1;
    }
    25      else
    {
        /* not enuf records skip past 1 or 2 bits */
        for ( i=0; i < j; i++)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
    30      }
    return(count);
}

int last_pur_prd_select(

    35      struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
)

```

```

{
    PUR_PRD_SEL_VARS;

    PUR_PRD_SEL_PRELIM;

    results = results_bitmap->start;

5   pur_prd_map = pur_prd_bitmap->start;
    pur_defn_map = next_pur_prd_bitmap->start;
    for(count=0; map < map_end; map++)
    {
        if ( (j = *map) >= 1 )
        {
            /* advance to last bit for this customer */
            for ( i=1; i<j; i++)
            {
10                NEXT_PUR_PRD_RESULT_BIT1;
            }

            while ( j > 0 )
            {
                /* looks at only items defined in next_pur_prd_bitmap for each customer */
                if (*pur_defn_map & *pur_prd_bits )
                {
                    /* now see item met the search criteria */
                    if ( *pur_prd_bits & *pur_prd_map )
15                {
                        count++;
                        *results |= *pur_prd_bits;
                    }
                    break;
                }
                PREV_PUR_PRD_RESULT_BIT;
            }
            for ( i,j < i,j++ )
20            {
                NEXT_PUR_PRD_RESULT_BIT1;
            }
        }
    }
    return(count);
}

25 int nil_pur_prd_select(
    struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
    )
{
    PUR_PRD_SEL_VARS;

    PUR_PRD_SEL_PRELIM;

30   results = results_bitmap->start;

    pur_prd_map = pur_prd_bitmap->start;
    pur_defn_map = next_pur_prd_bitmap->start;
    for(count=0; map < map_end; map++)
    {

35

```

```

    if ( (j = *map) >= 2 )
    {
        /* advance to last bit for this customer */
        for ( found=0, i=1; i<j; i++)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
5           }

        while ( j > 0 )
        {
            /* looks at only items defined in next_pur_prd_bitmap for each customer */
            if ( *pur_defn_map & *pur_prd_bits )
            {
                found++;
                if ( found == 2 )
10                {
                    /* now see item met the search criteria */
                    if ( *pur_prd_bits & *pur_prd_map )
                    {
                        count++;
                        *results |= *pur_prd_bits;
                    }
                    break;
                }
            }
            PREV_PUR_PRD_RESULT_BIT1;
15        }
        for ( j, i < 1; j++ )
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
    else
    {
        /* not enough records skip past 0 or 1 bits */
        for ( i=0; i < j; i++ )
20        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
}
return(count);
}

/* Note: this function could be done several ways initially set up to
25 select customers whose last 2 purchases BOTH met the search criteria
Revised so that select customers who had at least 1 of last 2 purchases
that met the criteria Now creating purchase bitmap marking all purchases
among the last 2 that met the criteria */

int last2_pur_prd_select(
30     struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
)
{
    PUR_PRD_SEL_VARS
    PUR_PRD_SEL_PRELIM.

    results = results_bitmap->start;

    pur_prd_map = pur_prd_bitmap->start;
    pur_defn_map = next_pur_prd_bitmap->start;
35    for(count=0; map < map_end; map++)
    {

```

```

    if ( (j = *map) >= 2 )
    {
        /* advance to last bit for this customer */
        for ( current = 0, found = 0, i=1; i<j; i++)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }

        while ( j > 0 )
        {
            /* looks at only items defined in next_pur_prd_bimap for each customer */
            if ( *pur_defn_map & *pur_prd_bits )
            {
                current++;
                /* now see item met the search criteria */
                if ( *pur_prd_bits & *pur_prd_map )
                {
                    count++;
                    *results |= *pur_prd_bits;
                    break;
                }
                if ( current >= 2 )
                    break;
            }
            PREV_PUR_PRD_RESULT_BIT;
        }
        for ( ; j < 1; j++)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
    else
    {
        /* not enuf records skip past 0, or 1 bits */
        for ( i=0; i < j; i++)
        {
            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
}
return(count);
}

/* Note: this function could be done several ways. Initially set up to
select customers whose last 5 purchases ALL met the search criteria
Revised so that select customers who had at least 1 of last 5 purchases
that met the criteria. Now creating purchase bitmap marking all purchases
among the last 5 that met the criteria */
int last5_pur_prd_select(
    struct bitmap *pur_prd_bitmap,
    struct bitmap *next_pur_prd_bitmap,
    enum boolean_operator oper,
    struct bitmap *results_bitmap
)
{
    PUR_PRD_SEL_VARS;

    PUR_PRD_SEL_PRELIM;

    results = results_bitmap->start;

    pur_prd_map = pur_prd_bitmap->start;
    pur_defn_map = next_pur_prd_bitmap->start;
    for(count=0; map < map_end; map++)
    {
        if ( (j = *map) >= 5 )
        {

```

```

    /* advance to test bit for this customer */
    for ( current = 0, found = 0, i=1; i<j; i++)
    {
        NEXT_PUR_PRD_RESULT_BIT1;
    }

5   while ( i > 0 )
    {
        /* looks at only items defined in next_pur_prd_bitmap for each customer */
        if ( *pur_dem_map & *pur_prd_bits )
        {
            current++;
            /* now see item met the search criteria */
            if ( *pur_prd_bits & *pur_prd_map )
            {
10                count++;
                *results |= *pur_prd_bits;
                break; /* found 1, stop looking now */
            }
            if ( current >= 5 )
                break;
        }
        PREV_PUR_PRD_RESULT_BIT;
    }
15   for ( j; j < i; j++)
    {
        NEXT_PUR_PRD_RESULT_BIT1;
    }
    else
    {
        /* not enuf records, skip past up to 4 bits */
        for ( i=0, i < j, i++)
        {
20            NEXT_PUR_PRD_RESULT_BIT1;
        }
    }
}
return(count);
}

25 int avg_pur_prd_large( unsigned int *data, unsigned int value, unsigned int high_value,
                        struct list_types *list, enum field_operator operator, int num_items, struct bitmap
                        *results_map, unsigned short *map, struct bitmap *pur_prd_bitmap )
{
    int denom, avg;
    unsigned int *data_end,
    unsigned int *sav_data,
    unsigned int *sav_bit,
    unsigned int *sav_results;

30   PUR_PRD_VARS;
    LIST_PRELIMS.

    switch ( operator )
    {
        case equal:
            PUR_PRD_AVG_PUR_PRD( if( avg == value ), MISSING_LARGE_VALUE );
            break;
35   case not_equal:
            PUR_PRD_AVG_PUR_PRD( if( avg != value ), MISSING_LARGE_VALUE );
            break;
    }

```



```

case equal_list:
pur_prd_map = pur_prd_bitmap->start;
for(count=0; map < map_end; map++)
{
    if ( *map >= 1 )
    {
        sav_data = data;
        sav_bit = pur_prd_bits;
        sav_results = results;
        data_end = data + *map;
        for ( total=0,denom=0, data < data_end; data++)
        {
            if(*data < MISSING_LARGE_VALUE)
            {
                denom++;
                total += *data;
            }
            NEXT_PUR_PRD_RESULT_BIT.
        }
        if ( denom )
            avg = (int)((float)total / (float)denom) + .5;
        else
            avg = 0;
        found = 0;

        list_head = list;
        while ( list_head != NULL )
        {
            if ( list_head->type_list == variable_value )
            {
                if ( avg == list_head->integer )
                {
                    found++;
                    break;
                }
            }
            else
            if ( list_head->type_list == range_value )
            {
                if ( (avg >= list_head->between_integer->low) &&
                    (avg <= list_head->between_integer->high) )
                {
                    found++;
                    break;
                }
            }
            list_head = list_head->next;
        }
        if ( found )
        {
            data = sav_data;
            pur_prd_bits = sav_bit;
            results = sav_results;
            data_end = data + *map;
            for ( ; data < data_end; data++)
            {
                *results |= *pur_prd_bits;
                count++;
            }
            NEXT_PUR_PRD_RESULT_BIT.
        }
    }
}

```

```

    )
    return(count);
    break;
    case not_equal_list:
5   pur_prd_map = pur_prd_bitmap->start;
    for(count=0; map < map_end; map++)
    {
        if ( *map >= 1 )
        {
            sav_data = data;
            sav_bit = pur_prd_bits;
            sav_results = results;
            data_end = data + *map;

10   for ( total=0; denom=0; data < data_end; data++)
        {
            if ( *data < MISSING_LARGE_VALUE )
            {
                denom++;
                total += *data;
            }

            NEXT_PUR_PRD_RESULT_BIT;

15   }

        if ( denom )
            avg = (int)((float)total / (float)denom + .5);
        else
            avg = 0;

        found = 0;
        list_head = list;
        while ( list_head != NULL )
        {
20   if ( list_head->type_list == variable_value )
            {
                if ( avg == list_head->integer )
                {
                    found++;
                    break;
                }
            }
            else
25   if ( list_head->type_list == range_value )
            {
                if ( (avg >= list_head->between_integer->low) &&
                    (avg <= list_head->between_integer->high) )
                {
                    found++;
                    break;
                }
            }
            list_head = list_head->next;

30   }
        if ( !found )
        {
            data = sav_data;
            pur_prd_bits = sav_bit;
            results = sav_results;
            data_end = data + *map;
            for ( ; data < data_end; data++)
            {
35   *results |= *pur_prd_bits;
                count++;
                NEXT_PUR_PRD_RESULT_BIT;
            }
        }
    }
}

```

```

    )
    return(count);
    break;
    case greater_than:
        PUR_PRD_AVG_PUR_PRD( if( avg > value ), MISSING_LARGE_VALUE );
        break;
    case less_than:
        PUR_PRD_AVG_PUR_PRD( if( avg < value ), MISSING_LARGE_VALUE );
        break;
    case greater_than_equal:
        PUR_PRD_AVG_PUR_PRD( if( avg >= value ), MISSING_LARGE_VALUE );
        break;
    case less_than_equal:
        PUR_PRD_AVG_PUR_PRD( if( avg <= value ), MISSING_LARGE_VALUE );
        break;
    case between:
        PUR_PRD_AVG_PUR_PRD( if( avg <= value && avg <= high_value ), MISSING_LARGE_VALUE );
        break;
    default:
        if (purchase_query)
            error_handler (PURCHASE_UNKNOWN_OP, ERROR, NO_STATUS, "avg_pur_prd_large");
        else
            error_handler (PRODUCT_UNKNOWN_OP, ERROR, NO_STATUS, "avg_pur_prd_large");
        break;
    )
}

int avg_pur_prd_medium( unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
                        enum field_operator operator, int num_items, struct bitmap *results_map,
                        unsigned short *map, struct bitmap *pur_prd_bitmap )
{
    int denom, avg;
    unsigned short *data_end;
    unsigned short *sav_data;
    unsigned int *sav_bit;
    unsigned int *sav_results;

    PUR_PRD_VARS;

    LIST_PRELIMS;

    switch ( operator )
    {
    case equal:
        PUR_PRD_AVG_PUR_PRD( if( avg == value ), MISSING_MEDIUM_VALUE );
        break;
    case not_equal:
        PUR_PRD_AVG_PUR_PRD( if( avg != value ), MISSING_MEDIUM_VALUE );
        break;
    case equal_list:
        pur_prd_map = pur_prd_bitmap->start;
        for(count=0; map < map_end; map++)
        {
            if ( *map >= 1 )
            {
                sav_data = data;
                sav_bit = pur_prd_bits;
                sav_results = results;
                data_end = data + *map;
                for ( total=0, denom=0, data < data_end; data++)
                {
                    if(*data < MISSING_MEDIUM_VALUE)
                }
            }
        }
    }
}

```

```

denom++;
total += *data;
}
NEXT_PUR_PRD_RESULT_BIT.
}
5 if ( denom )
    avg = (int)((float)total / (float)denom) * .5;
else
    avg = 0;
found = 0;
list_head = list;
while ( list_head != NULL )
{
10 if ( list_head->type_list == variable_value )
    {
        if ( avg == list_head->integer )
        {
            found++;
            break;
        }
    }
else
15 if ( list_head->type_list == range_value )
    {
        if ( (avg >= list_head->between_integer->low) &&
            (avg <= list_head->between_integer->high) )
        {
            found++;
            break;
        }
    }
    list_head = list_head->next;
}
20 if ( found )
{
    data = sav_data;
    pur_prd_bits = sav_bit;
    results = sav_results;
    data_end = data + *map;
    for ( ; data < data_end; data++ )
    {
        *results |= *pur_prd_bits;
        count++;
25 NEXT_PUR_PRD_RESULT_BIT.
    }
}
}
return(count);
break;
case not_equal_list
pur_prd_map = pur_prd_bitmap->start;
for(count=0; map < map_end; map++)
30 {
    if ( *map >= 1 )
    {
        sav_data = data;
        sav_bit = pur_prd_bits;
        sav_results = results;
        data_end = data + *map;
        for ( total=0,denom=0; data < data_end; data++ )
        {
35 if(*data < MISSING_MEDIUM_VALUE)

```

```

    {
        denom++;
        total += *data;
    }
    NEXT_PUR_PRD_RESULT_BIT;
}

if ( denom )
    avg = (int)((float)total / (float)denom) + .5;
else
    avg = 0;

found = 0;
list_head = list;
while ( list_head != NULL )
{
    if ( list_head->type_list == variable_value )
    {
        if ( avg == list_head->integer )
        {
            found++;
            break;
        }
    }
    else
    {
        if ( list_head->type_list == range_value )
        {
            if ( (avg >= list_head->between_integer->low) &&
                (avg <= list_head->between_integer->high) )
            {
                found++;
                break;
            }
        }
        list_head = list_head->next;
    }
}

if ( !found )
{
    data = sav_data;
    pur_prd_bits = sav_bit;
    results = sav_results;
    data_end = data + *map;
    for ( ; data < data_end; data++ )
    {
        *results |= *pur_prd_bits;
        count++;
    }
    NEXT_PUR_PRD_RESULT_BIT;
}
}

return(count);
break;
case greater_than
    PUR_PRD_AVG_PUR_PRD( if( avg > value ), MISSING_MEDIUM_VALUE );
    break;
case less_than
    PUR_PRD_AVG_PUR_PRD( if( avg < value ), MISSING_MEDIUM_VALUE );
    break;
case greater_than_equal
    PUR_PRD_AVG_PUR_PRD( if( avg >= value ), MISSING_MEDIUM_VALUE );
    break;
case less_than_equal
    PUR_PRD_AVG_PUR_PRD( if( avg <= value ), MISSING_MEDIUM_VALUE );
    break;
case between
    PUR_PRD_AVG_PUR_PRD( if( avg <= value && avg <= high_value ), MISSING_MEDIUM_VALUE );
    break;
default
    if (purchase_query)
        error_handler (PURCHASE_UNKNOWN_OP, ERROR, NO_STATUS, "avg_pur_prd_medium");
    else
        error_handler (PRODUCT_UNKNOWN_OP, ERROR, NO_STATUS, "avg_pur_prd_medium");
    break;
}
}

```

```

int avg_pur_prd_date( unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
5 { unsigned short *map, struct bitmap *pur_prd_bitmap )
{
    error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Average Purchase/Product for a date is not available ");
}

int avg_pur_prd_short( unsigned char *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
10 { unsigned short *map, struct bitmap *pur_prd_bitmap )
{
    int denom, avg;
    unsigned char *data_end;
    unsigned char *sav_data;
    unsigned int *sav_bit;
    unsigned int *sav_results;

    PUR_PRD_VARS;
15 LIST_PRELIMS;

    switch ( operator )
    {
        case equal:
            PUR_PRD_AVG_PUR_PRD( if( avg == value ), MISSING_SHORT_VALUE );
            break;
        case not_equal:
            PUR_PRD_AVG_PUR_PRD( if( avg != value ), MISSING_SHORT_VALUE );
            break;
20 case equal_list:
            pur_prd_map = pur_prd_bitmap->start;
            for(count=0; map < map_end; map++)
            {
                if ( *map >= 1 )
                {
                    sav_data = data;
                    sav_bit = pur_prd_bits;
                    sav_results = results;
                    data_end = data + *map;
25 for ( total=0, denom=0, data < data_end, data++)
                    {
                        if(*data < MISSING_SHORT_VALUE)
                        {
                            denom++;
                            total += *data;
                        }
                    }
                    NEXT_PUR_PRD_RESULT_BIT;
30 if ( denom )
                        avg = (int)((float)total / (float)denom) + 5);
                    else
                        avg = 0;
                    found = 0;
                    list_head = list;
                    while ( list_head != NULL )
                    {
                        if ( list_head->type_list == variable_value )
                        {
                            if ( avg == list_head->integer )
                            {
                                found++;
                                break;
35
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    else
    if ( list_head->type_list == range_value )
    {
        if ( (avg >= list_head->between_integer->low) &&
            (avg <= list_head->between_integer->high) )
        {
            found++;
            break;
        }
    }
    list_head = list_head->next;
}
if ( found )
{
    data = sav_data;
    pur_pro_bits = sav_bit;
    results = sav_results;
    data_end = data + *map;
    for ( ; data < data_end; data++)
    {
        *results |= *pur_pro_bits;
        count++;
    }
    NEXT_PUR_PRD_RESULT_BIT;
}
}
return(count);
break;
case not_equal_list
pur_pro_map = pur_pro_bitmap->start;
for(count=0; map < map_end; map++)
{
    if ( *map >= 1 )
    {
        sav_data = data;
        sav_bit = pur_pro_bits;
        sav_results = results;
        data_end = data + *map;
        for ( total=0, denom=0, data < data_end; data++ )
        {
            if(*data < MISSING_SHORT_VALUE)
            {
                denom++;
                total += *data;
            }
            NEXT_PUR_PRD_RESULT_BIT;
        }
        if ( denom )
            avg = (int)((float)total / (float)denom) + .5;
        else
            avg = 0;

        found = 0;
        list_head = list;
        while ( list_head != NULL )
        {
            if ( list_head->type_list == variable_value )
            {
                if ( avg == list_head->integer )
                {
                    found++;
                    break;
                }
            }
        }
    }
}

```

```

    }
    }
    else
    {
        if ( list_head->type_list == range_value )
        {
            if ( (avg >= list_head->between_integer->low) &&
                (avg <= list_head->between_integer->high) )
            {
                found++;
                break;
            }
        }
        list_head = list_head->next;
    }
    if ( !found )
    {
        data = sav_data;
        pur_prd_bits = sav_bit;
        results = sav_results;
        data_end = data + *map;
        for ( ; data < data_end; data++)
        {
            *results |= *pur_prd_bits;
            count++;
        }
        NEXT_PUR_PRD_RESULT_BIT;
    }
}
return(count);
break;
case greater_than
PUR_PRD_AVG_PUR_PRD( if( avg > value ) MISSING_SHORT_VALUE );
break;
case less_than
PUR_PRD_AVG_PUR_PRD( if( avg < value ) MISSING_SHORT_VALUE );
break;
20 case greater_than_equal
PUR_PRD_AVG_PUR_PRD( if( avg >= value ) MISSING_SHORT_VALUE );
break;
case less_than_equal
PUR_PRD_AVG_PUR_PRD( if( avg <= value ) MISSING_SHORT_VALUE );
break;
case between
25 PUR_PRD_AVG_PUR_PRD( if( avg <= value && avg <= high_value ) MISSING_SHORT_VALUE );
break;
default
if (purchase_query)
    error_handler (PURCHASE_UNKNOWN_OP, ERROR_NO_STATUS "avg_pur_pro_short");
else
    error_handler (PRODUCT_UNKNOWN_OP, ERROR_NO_STATUS "avg_pur_pro_short");
break;
}
}
30
int avg_pur_pro_char( char *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_pro_bitmap )
{
    error_handler (FUNC_NOT_AVAIL, ERROR_NO_STATUS, "Average Purchase/Product for a character
is not available.");
}
35

```



```

int avg_pur_prd_string(char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
5   error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Average Purchase/Product for a string is not available ");
}

int avg_pur_prd_mixed_string(char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
10  error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Average Purchase/Product for a mixed string is not
available.");
}

int avg_pur_prd_fstring(unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
15  error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Average Purchase/Product for a fixed string is not
available.");
}

int avg_pur_prd_mixed_fstring(unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
20  error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Average Purchase/Product for a mixed fixed string is not
available ");
}

int avg_pur_prd_bit( int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Average Purchase/Product for a bit is not available ");
}

int total_pur_prd_large( unsigned int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
25  unsigned int *data_end;
unsigned int *sav_data;
unsigned int *sav_bit;
unsigned int *sav_results;

PUR_PRD_VARS;
30  LIST_PRELIMS.

switch ( operator )
{
case equal:
PUR_PRD_TOTAL_PUR_PRD( if( total == value), MISSING_LARGE_VALUE );
break;
case not_equal:
PUR_PRD_TOTAL_PUR_PRD( if( total != value), MISSING_LARGE_VALUE );
break;
35  case equal_list:
PUR_PRD_LOOP_PART_1( );
PUR_PRD_LOOP_GET_TOTAL( MISSING_LARGE_VALUE );
found = 0;
list_head = list;
while ( list_head != NULL )
{

```

100

```

    if ( list_head->type_list == variable_value )
    {
        if ( total == list_head->integer )
        {
            found++;
            break;
        }
        else
        {
            if ( list_head->type_list == range_value )
            {
                if ( (total >= list_head->between_integer->low) &&
                    (total <= list_head->between_integer->nigh) )
                {
                    found++;
                    break;
                }
            }
        }
        list_head = list_head->next;
    }
    if ( found )
        PUR_PRD_LOOP_END_TOTAL;
    break;
case not_equal_list:
    PUR_PRD_LOOP_PART_1( );
    PUR_PRD_LOOP_GET_TOTAL( MISSING_LARGE_VALUE );
    found = 0;
    list_head = list;
    while ( list_head != NULL )
    {
        if ( list_head->type_list == variable_value )
        {
            if ( total == list_head->integer )
            {
                found++;
                break;
            }
        }
        else
        {
            if ( list_head->type_list == range_value )
            {
                if ( (total >= list_head->between_integer->low) &&
                    (total <= list_head->between_integer->nigh) )
                {
                    found++;
                    break;
                }
            }
        }
        list_head = list_head->next;
    }
    if ( found )
        PUR_PRD_LOOP_END_TOTAL;
    break;
case greater_than:
    PUR_PRD_TOTAL_PUR_PRD( if( total > value), MISSING_LARGE_VALUE );
    break;
case less_than:
    PUR_PRD_TOTAL_PUR_PRD( if( total < value), MISSING_LARGE_VALUE );
    break;
case greater_than_equal:
    PUR_PRD_TOTAL_PUR_PRD( if( total >= value), MISSING_LARGE_VALUE );
    break;
case less_than_equal:
    PUR_PRD_TOTAL_PUR_PRD( if( total <= value), MISSING_LARGE_VALUE );
    break;
case between:
    PUR_PRD_TOTAL_PUR_PRD( if( total >= value && total <= high_value), MISSING_LARGE_VALUE );
    break;
default:
    if (purchase_query)
        error_handler( PURCHASE_UNKNOWN_OP, ERROR_NO_STATUS, "total_pur_prd_large");

```

```

        else
            error_handler (PRODUCT_UNKNOWN_OP, ERROR_NO_STATUS, "total_pur_prd_large")
        break;
    )
5 )

int total_pur_prd_medium( unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
    enum field_operator operator, int num_items, struct bitmap *results_map,
    unsigned short *map, struct bitmap *pur_prd_bitmap )
{
    unsigned short *data_end;
    unsigned short *sav_data;
    unsigned int *sav_bit;
10 unsigned int *sav_results;

    PUR_PRD_VARS;
    LIST_PRELIMS;

    switch ( operator )
    {
        case equal:
15 PUR_PRD_TOTAL_PUR_PRD( if( total == value), MISSING_MEDIUM_VALUE );
            break;
        case not_equal:
            PUR_PRD_TOTAL_PUR_PRD( if( total != value), MISSING_MEDIUM_VALUE );
            break;
        case equal_list:
            PUR_PRD_LOOP_PART_1( );
            PUR_PRD_LOOP_GET_TOTAL( MISSING_MEDIUM_VALUE );
            found = 0;
            list_head = list;
20 while ( list_head != NULL )
            {
                if ( list_head->type_list == variable_value )
                {
                    if ( total == list_head->integer )
                    {
                        found++;
                        break;
                    }
25 else
                    if ( list_head->type_list == range_value )
                    {
                        if ( (total >= list_head->between_integer->low) &&
                            (total <= list_head->between_integer->high) )
                        {
                            found++;
                            break;
                        }
                    }
30 }
                list_head = list_head->next;
            }
            if ( found )
                PUR_PRD_LOOP_END_TOTAL;

            break;
        case not_equal_list:
            PUR_PRD_LOOP_PART_1( );
            PUR_PRD_LOOP_GET_TOTAL( MISSING_MEDIUM_VALUE );
35 found = 0;
            list_head = list;
            while ( list_head != NULL )
            {

```

```

    if ( list_head->type_list == variable_value )
    {
        if ( total == list_head->integer )
        {
            found++;
            break;
        }
        else
        if ( list_head->type_list == range_value )
        {
            if ( (total >= list_head->between_integer->low) &&
                (total <= list_head->between_integer->high) )
            {
                found++;
                break;
            }
        }
        list_head = list_head->next;
    }
    if ( !found )
        PUR_PRD_LOOP_END_TOTAL;

    break;
case greater_than:
15   PUR_PRD_TOTAL_PUR_PRD( if( total > value), MISSING_MEDIUM_VALUE );
    break;
case less_than:
    PUR_PRD_TOTAL_PUR_PRD( if( total < value), MISSING_MEDIUM_VALUE );
    break;
case greater_than_equal:
    PUR_PRD_TOTAL_PUR_PRD( if( total >= value), MISSING_MEDIUM_VALUE );
    break;
case less_than_equal:
20   PUR_PRD_TOTAL_PUR_PRD( if( total <= value), MISSING_MEDIUM_VALUE );
    break;
case between:
    PUR_PRD_TOTAL_PUR_PRD( if( total >= value && total <= high_value), MISSING_MEDIUM_VALUE );
    break;
default:
    if (purchase_query)
        error_handler (PURCHASE_UNKNOWN_OP, ERROR, NO_STATUS, "total_pur_prd_medium");

    else
25       error_handler (PRODUCT_UNKNOWN_OP, ERROR, NO_STATUS, "total_pur_prd_medium");
    break;
}
}

int total_pur_prd_date( unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
30   enum field_operator operator, int num_rems, struct bitmap *results_map,
    unsigned short *map, struct bitmap *pur_prd_bitmap )
{
    error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Total Purchase/Product for a date is not available.");
}

int total_pur_prd_short( unsigned char *data, unsigned int value, unsigned int high_value, struct list_types *list,
35   enum field_operator operator, int num_rems, struct bitmap *results_map,
    unsigned short *map, struct bitmap *pur_prd_bitmap )

```

```

unsigned char *data_end;
unsigned char *sav_data;
unsigned int *sav_ptr;
unsigned int *sav_results;

5  PUR_PRD_VARS;

LIST_PRELIMS;

switch ( operator )
{
    case equal:
        PUR_PRD_TOTAL_PUR_PRD( if( total == value), MISSING_SHORT_VALUE );
        break;
10  case not_equal:
        PUR_PRD_TOTAL_PUR_PRD( if( total != value), MISSING_SHORT_VALUE );
        break;
    case equal_list:
        PUR_PRD_LOOP_PART_1( );
        PUR_PRD_LOOP_GET_TOTAL( MISSING_SHORT_VALUE );
        found = 0;
        list_head = list;
        while ( list_head != NULL )
15  {
            if ( list_head->type_list == variable_value )
            {
                if ( total == list_head->integer )
                {
                    found++;
                    break;
                }
            }
            else
            if ( list_head->type_list == range_value )
20  {
                if ( (total >= list_head->between_integer->low) &&
                    (total <= list_head->between_integer->high) )
                {
                    found++;
                    break;
                }
            }
            list_head = list_head->next;
25  }
        if ( found )
            PUR_PRD_LOOP_END_TOTAL;
        break;
    case not_equal_list:
        PUR_PRD_LOOP_PART_1( );
        PUR_PRD_LOOP_GET_TOTAL( MISSING_SHORT_VALUE );
        found = 0;
        list_head = list;
30  while ( list_head != NULL )
        {
            if ( list_head->type_list == variable_value )
            {
                if ( total == list_head->integer )
                {
                    found++;
                    break;
                }
            }
            else
            if ( list_head->type_list == range_value )
35  {

```

```

        if ( (total >= list_head->between_miege->low) &&
              (total <= list_head->between_miege->high) )
        {
            found++;
            break;
        }
    }
    list_head = list_head->next;
}
if ( found )
    PUR_PRD_LOOP_END_TOTAL;
break;
case greater_than:
    PUR_PRD_TOTAL_PUR_PRD( if( total > value), MISSING_SHORT_VALUE );
    break;
10 case less_than:
    PUR_PRD_TOTAL_PUR_PRD( if( total < value), MISSING_SHORT_VALUE );
    break;
case greater_than_equal:
    PUR_PRD_TOTAL_PUR_PRD( if( total >= value), MISSING_SHORT_VALUE );
    break;
case less_than_equal:
    PUR_PRD_TOTAL_PUR_PRD( if( total <= value), MISSING_SHORT_VALUE );
    break;
15 case between:
    PUR_PRD_TOTAL_PUR_PRD( if( total >= value && total <= high_value), MISSING_SHORT_VALUE );
    break;
default:
    if (purchase_query)
        error_handler (PURCHASE_UNKNOWN_OP, ERROR, NO_STATUS, "total_pur_prd_short");
    else
        error_handler (PRODUCT_UNKNOWN_OP, ERROR, NO_STATUS, "total_pur_prd_short");
20 break;
}
}

int total_pur_prd_char( char *data, unsigned int value, unsigned int high_value, struct list_types *list,
                      enum field_operator operator, int num_rems, struct bitmap *results_map,
                      unsigned short *map, struct bitmap *pur_prd_bitmap )
25 {
    error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Total Purchase/Product for a character is not available.");
}

int total_pur_prd_string(char *data, char *search_string, int str_len, struct list_types *list,
                       enum field_operator operator, int num_rems, struct bitmap *results_map,
                       unsigned short *map, struct bitmap *pur_prd_bitmap )
30 {
    error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Total Purchase/Product for a string is not available.");
}

int total_pur_prd_mixed_string(char *data, char *search_string, int str_len, struct list_types *list,
                              enum field_operator operator, int num_rems, struct bitmap *results_map,
                              unsigned short *map, struct bitmap *pur_prd_bitmap )
35 {
    error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Total Purchase/Product for a mixed string is not available.");
}

```

```

int total_pur_prd_fstring(unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap)
5 {
    error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Total Purchase/Product for a fixed string is not available.");
}

int total_pur_prd_mixed_fstring(unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap)
{
    error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Total Purchase/Product for a mixed fixed string
10 is not available.");
}

int total_pur_prd_bit( int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap)
{
    error_handler (FUNC_NOT_AVAIL, ERROR, NO_STATUS, "Total Purchase/Product for a bit is not available.");
}

/* As of May 1993, N_pur_prd_bitmap is not supported yet and there are no definite plans to add this. This operator requires
15 an additional parameter (Value N - spelling out how many records to use). This requires changes to the parser and
search section in addition to adding software here to support this. Also routine PUR_PRD_QUERY in this module */
int n_pur_prd_large( unsigned int *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_rems, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap)
{
    unsigned int *data_end;
    PUR_PRD_VARS;
20 LIST_PRELIMS;
    switch ( operator )
    {
        case equal:
        case not_equal:
        case equal_list:
        case not_equal_list:
        case greater_than:
25 case less_than
        case greater_than_equal
        case less_than_equal
        case between
        default:
            return(0);
    }
    /* djl - to be added. Not sure what to do here 1st n or last n, where n is passed in argument list.
30 for(count=0; map < map_end; map++)
    {
        for (total=0; data < data + *map ;data++)
        {
            if(*pur_prd_bits & *pur_prd_map)
            {
                total++;
            }
            NEXT_PUR_PRD_BIT;
        }
        if(total == target)
35 {
            *results = *results | *bits;
            count++;
        }
        NEXT_RESULT_BIT;
    }
    return(count);

```

```

/* dan, since this functionality is not implemented, I am returning void. */
void n_pur_prd_medium( unsigned short *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
5 {
/* Possible future expansion of functionality. When we actually support this, the above N_PUR_PRD_LARGE will be
copied to here and the data type revised. No current plans for this. */
}

/* dan, since this functionality is not implemented, I am returning void. */
10 void n_pur_prd_short( unsigned char *data, unsigned int value, unsigned int high_value, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
/* Possible future expansion of functionality. When we actually support this, the above N_PUR_PRD_LARGE will be
copied to here and the data type revised. No current plans for this. */
}

15 /* dan, since this functionality is not implemented, I am returning void. */
void n_pur_prd_char( char *data, int value, int high_value, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
/* Possible future expansion of functionality. When we actually support this, the above N_PUR_PRD_LARGE will be
copied to here and the data type revised. No current plans for this. */
}

/* dan, since this functionality is not implemented, I am returning void. */
20 void n_pur_prd_string( char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
/* Possible future expansion of functionality. When we actually support this, the above N_PUR_PRD_LARGE will be
copied to here and the data type revised. No current plans for this. */
}

/* dan, since this functionality is not implemented, I am returning void. */
25 void n_pur_prd_mixed_string( char *data, char *search_string, int str_len, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
/* Possible future expansion of functionality. When we actually support this, the above N_PUR_PRD_LARGE will be
copied to here and the data type revised. No current plans for this. */
}

/* dan, since this functionality is not implemented, I am returning void. */
30 void n_pur_prd_tstring( unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
/* Possible future expansion of functionality. When we actually support this, the above N_PUR_PRD_LARGE will be
copied to here and the data type revised. No current plans for this. */
}

/* dan, since this functionality is not implemented, I am returning void. */
35 void n_pur_prd_mixed_tstring( unsigned short *data, char *search_string, int str_len, int field_offset, struct list_types *list,
enum field_operator operator, int num_items, struct bitmap *results_map,
unsigned short *map, struct bitmap *pur_prd_bitmap )
{
/* Possible future expansion of functionality. When we actually support this, the above N_PUR_PRD_LARGE will be
copied to here and the data type revised. No current plans for this. */
}

```



```

/* dan, since this functionality is not implemented, I am returning void */
void n_pur_prd_bit( int *data, unsigned int value, unsigned int high_value, struct list_types *list,
                  enum field_operator operator, int num_rems, struct bitmap *results_map,
                  unsigned short *map, struct bitmap *pur_prd_bitmap )
{
5   /* Possible future expansion of functionality. When we actually support this, the above N_PUR_PRD_LARGE will be
   copied to here and the data type revised. No current plans for this. */
}

/*
-- PUR_PRD_QUERY ROUTINE
--
*/

10 int pur_prd_query_special ( struct dataset *data,
                             enum data_type field_type,
                             int field_size,
                             int field_offset,
                             enum field_operator operator,
                             union search_item search_value,
                             enum search_value_item search_value_type,
                             enum pur_prd_query_type query_type,
                             struct bitmap *results_bitmap)
15 {
    unsigned int total_count = 0;
    int reference_count;
    int num_bits;
    unsigned short *pur_prd_map;
    struct bitmap *pur_prd_bitmap;

    if (purchase_query)
20     {
        pur_prd_map = pur01_map;
        pur_prd_bitmap = pur01_bitmap;
    }
    else
    {
        pur_prd_map = prd01_map;
        pur_prd_bitmap = prd01_bitmap;
    }

25     num_bits = pur01_bitmap->number_of_bits;

    if (query_type == avg_pur_prd)
        reference_count = 0
    if (query_type == total_pur_prd)
        reference_count = 1

    switch (field_type)
    {
30         case dollars:
        case floating_point:
        case large_integer:
            if (search_value_type == list_value)
                /* we are looking at a list of values */
            {
                total_count = (*pur_prd_large_special)(reference_count)(
                                                                    data->items,
                                                                    (unsigned int)0,
                                                                    (unsigned int)0,
                                                                    search_value list,
                                                                    operator,
                                                                    num_bits,
                                                                    results_bitmap,
                                                                    pur_prd_map,
                                                                    pur_prd_bitmap );
35
            }
        }
    }
}

```

```

    }
    else
    {
        if (operator==between)
            total_count = (*pur_prd_large_special|reference_count));
            data->items
            search_value.between_integer->low
            search_value.between_integer->high
            NULL
            operator
            num_bits
            results_bitmap
            pur_prd_map
            pur_prd_bitmap );

        else
            total_count = (*pur_prd_large_special|reference_count));
            data->items
            search_value.integer
            (unsigned int)0
            NULL
            operator
            num_bits
            results_bitmap
            pur_prd_map
            pur_prd_bitmap );

    }
    break;
case negative_float:
    if (search_value_type == list_value)
        /* we are looking at a list of values */

    {
        total_count = (*pur_prd_large_special|reference_count));
            data->items
            (int)0
            (int)0
            search_value.list
            operator
            num_bits
            results_bitmap

            pur_prd_map
            pur_prd_bitmap );

    }
    else
    {
        if (operator==between)
            total_count = (*pur_prd_large_special|reference_count));
            data->items
            search_value.between_integer->low
            search_value.between_integer->high
            NULL
            operator
            num_bits
            results_bitmap
            pur_prd_map
            pur_prd_bitmap );

        else
            total_count = (*pur_prd_large_special|reference_count));
            data->items
            search_value.integer
            (int)0
            NULL
            operator
            num_bits
            results_bitmap
            pur_prd_map
            pur_prd_bitmap );
    }
}

```

```

    )
    break;
case year_month:
case year_month_day:
    if (search_value_type == list_value)
        /* we are looking at a list of values */
5      {
          total_count = (*pur_prd_date_special)(reference_count){
                                                    data->items,
                                                    (unsigned int)0,
                                                    (unsigned int)0,
                                                    search_value.list,
                                                    operator,
                                                    num_bits,
                                                    results_bitmap,
                                                    pur_prd_map,
                                                    pur_prd_bitmap );
10      }
    else
    {
        if (operator==between)
            total_count = (*pur_prd_date_special)(reference_count){
                                                    data->items,
                                                    search_value.between_integer->low,
15          search_value.between_integer->high,
                                                    NULL,
                                                    operator,
                                                    num_bits,
                                                    results_bitmap,
                                                    pur_prd_map,
                                                    pur_prd_bitmap );
        else
            total_count = (*pur_prd_date_special)(reference_count){
                                                    data->items,
20          search_value.integer,
                                                    (unsigned int)0,
                                                    NULL,
                                                    operator,
                                                    num_bits,
                                                    results_bitmap,
                                                    pur_prd_map,
                                                    pur_prd_bitmap );
    }
    break;
case medium_integer
25  if (search_value_type == list_value)
        /* we are looking at a list of values */
    {
        total_count = (*pur_prd_medium_special)(reference_count){
                                                    data->items,
                                                    (unsigned int)0,
                                                    (unsigned int)0,
30          search_value.list,
                                                    operator,
                                                    num_bits,
                                                    results_bitmap,
                                                    pur_prd_map,
                                                    pur_prd_bitmap );
    }
    else
    {
        if (operator==between)
            total_count = (*pur_prd_medium_special)(reference_count){
                                                    data->items,
35          search_value.between_integer->low,
                                                    search_value.between_integer->high,
                                                    NULL,
                                                    operator,
                                                    num_bits,
                                                    results_bitmap,
                                                    pur_prd_map,
                                                    pur_prd_bitmap );
        else
    }

```

```

total_count = (*pur_pro_medium_special(reference_count))(
    data->items,
    search_value integer,
    (unsigned int)0,
    NULL,
    operator,
    num_bits,
    results_bitmap,
    pur_pro_map,
    pur_pro_bitmap );

5
    }
    break;
case character:
case small_integer:
10
    if (search_value_type == list_value)
        /* we are looking at a list of values */
        {
            total_count = (*pur_pro_short_special(reference_count))(
                data->items,
                (unsigned int)0,
                (unsigned int)0,
                search_value.list,
                operator,
                num_bits,
                results_bitmap,
                pur_pro_map,
                pur_pro_bitmap );

15
        }
    else
    {
        if (operator==between)
        20
            total_count = (*pur_pro_short_special(reference_count))(
                data->items,
                search_value.between_integer->low,
                search_value.between_integer->high,
                NULL,
                operator,
                num_bits,
                results_bitmap,
                pur_pro_map,
                pur_pro_bitmap );

        else
        25
            total_count = (*pur_pro_short_special(reference_count))(
                data->items,
                search_value integer,
                (unsigned int)0,
                NULL,
                operator,
                num_bits,
                results_bitmap,
                pur_pro_map,
                pur_pro_bitmap );

30
    }
    break;
case fixed_string
    if (search_value_type == list_value)
        /* we are looking at a list of values */
        {
            /* see if any mixed strings or not */
            if ( check_list_for_mixed_strings( search_value.list ) )
            {
                /* mixed strings case */
                total_count = (*pur_pro_mixed_string_special(reference_count))(
                    data->items,
                    search_value.string->string_value,
                    search_value.string->string_length,

35
                    field_offset,
                    search_value.list,

```

111

```

operator,
results_bitmap,

num_bits
pur_prd_map,
pur_prd_bitmap );

5      )
      else
      {
        /* strings without ? or * wildcard characters */
        total_count = (*pur_prd_fstring_special(reference_count))(
data->items,
search_value.string->string_value,
search_value.string->string_length,

10      operator,
results_bitmap,

field_offset,
search_value.list,

num_bits,

pur_prd_map,
pur_prd_bitmap );

      )
    }
    else
    {
15      if (((strchr(search_value.string->string_value, '?')) == NULL) &&
        ((strchr(search_value.string->string_value, '*')) == NULL))
        total_count = (*pur_prd_fstring_special(reference_count))(
data->items,
search_value.string->string_value,
search_value.string->string_length,

        operator,
        NULL,

        num_bits,

20      results_bitmap,

        pur_prd_map,
        pur_prd_bitmap );

        else
        {
          total_count = (*pur_prd_mixed_fstring_special(reference_count))(
data->items,
search_value.string->string_value,
search_value.string->string_length,

        field_offset,
        NULL,

        num_bits,

25      operator,
results_bitmap,

        pur_prd_map,
        pur_prd_bitmap );

        )
      }
    }
    break;
    case string_type:
    {
30      if (search_value_type == list_value)
        /* we are looking at a list of values */
        {
          /* see if any mixed strings or not */
          if ( check_list_for_mixed_strings( search_value.list ) )
          {
            /* mixed strings case */
            total_count = (*pur_prd_mixed_string_special(reference_count))(
data->items,
search_value.string->string_value,
search_value.string->string_length,

35      operator,
results_bitmap,

search_value.list,

num_bits,

pur_prd_map,
pur_prd_bitmap );
          }
        }
    }
  }
}

```

```

    }
    else
    {
        /* strings without ? or * wildcard characters */
        total_count = (*pur_prd_string_special|reference_count));
data->items,
search_value.string->string_value,
5 search_value.string->string_length,
                                search_value list,
                                num_bits,
                                results_bitmap,
                                pur_prd_map,
                                pur_prd_bitmap );
    }
}
else
10 {
    if (((strchr(search_value.string->string_value, "?") == NULL) &&
        ((strchr(search_value.string->string_value, "*") == NULL)))
        total_count = (*pur_prd_string_special|reference_count));
data->items,
search_value string->string_value,
search_value string->string_length,
NULL,
operator,
15 num_bits,
results_bitmap,
pur_prd_map,
pur_prd_bitmap );
    else
        total_count = (*pur_prd_mixed_string_special|reference_count));
data->items,
search_value string->string_value,
search_value string->string_length,
NULL,
operator,
20 num_bits,
results_bitmap,
pur_prd_map,
pur_prd_bitmap );
}
break;
case bit_type:
    if (search_value_type == list_value)
        /* we are looking at a list of values */
25 {
            total_count = (*pur_prd_bit_special|reference_count));
data->items,
(unsigned int)0,
(unsigned int)0,
search_value list,
operator,
30 num_bits,
results_bitmap,
pur_prd_map,
pur_prd_bitmap );
        }
    else
    {
        total_count = (*pur_prd_bit_special|reference_count));
data->items,
35

```

```

5
    }
    }
    break;
    case bad_data_type:
    default: break;
} /* end of switch */

10 return(total_count);
}

/*
 * Not being used.
 */
void pur_pro_query (
    enum pur_pro_query_type query_type,
    struct bitmap *results_bitmap)
15 {
    unsigned int total_count = 0, temp;
    struct bitmap *query_bitmap;

    if (purchase_query && (master_purchase_bitmap == NULL) &&
        (query_type != pur_pro_domain))
    {
        if ((query_bitmap = create_general_bitmap(results_bitmap->number_of_bits)) == NULL)
            error_handler (BITMAP_NOT_CREATE, ERROR, NO_STATUS, "purchase bitmap in pur_pro_query");
20 #ifdef DEBUG
        fprintf(debug_file, "%s\n", "PUR_PRO_QUERY", "RESULTS_BITMAP", results_bitmap, "Mallocated");
        fprintf(debug_file, "%s\n", "PUR_PRO_QUERY", "RESULTS_BITMAP->START", results_bitmap->start,
            "Mallocated");
        fprintf(debug_file, "%s\n", "PUR_PRO_QUERY", "RESULTS_BITMAP->END", results_bitmap->end,
            "Mallocated");
        sendit

        if (copy_bitmaps(query_bitmap, results_bitmap) == NULL)
            error_handler (BITMAP_NOT_COPY, ERROR, NO_STATUS,
25 "results_bitmap to query_bitmap in pur_pro_query");

        for (temp=results_bitmap->start; temp<=results_bitmap->end; temp++)
            temp = 0;

        total_count = (*pur_pro_select[query_type])

30
    }
    else
    {
        if (product_query && (master_product_bitmap == NULL) &&
            (query_type != pur_pro_domain))
        {
            if ((query_bitmap = create_general_bitmap(results_bitmap->number_of_bits)) == NULL)
                error_handler (BITMAP_NOT_CREATE, ERROR, NO_STATUS, "purchase bitmap in pur_pro
35 _query");
            #ifdef DEBUG
            fprintf(debug_file, "%s\n", "PUR_PRO_QUERY", "RESULTS_BITMAP", results_bitmap, "Mallocated");
            fprintf(debug_file, "%s\n", "PUR_PRO_QUERY", "RESULTS_BITMAP->START", results_bitmap->start,
                "Mallocated");
            fprintf(debug_file, "%s\n", "PUR_PRO_QUERY", "RESULTS_BITMAP->END", results_bitmap->end,
                "Mallocated");
            sendit

```

```

(unsigned int)0,
(unsigned int)0,
NULL,
operator,
num_bits,
results_bitmap,
pur_pro_map,
pur_pro_bitmap ).

```

```

query_bitmap,
pur01_bitmap,
and,
results_bitmap );

```

```

    if (copy_bitmaps(query_bitmap, results_bitmap) == NULL)
        error_handler (BITMAP_NOT_COPY, ERROR, NO_STATUS,
            "results_bitmap to query_bitmap in pur_prd_query");
5
    for ( temp=results_bitmap->start; temp<=results_bitmap->end; temp++ )
        temp = 0;

    total_count = (*pur_prd_select(query_type))(

                                                                    query_bitmap,
                                                                    prd01_bitmap,
                                                                    and,
                                                                    results_bitmap );
10
    )
}

int combine_pur_prd_bitmap (
                                                                    enum pur_prd_query_type query_type,
                                                                    struct bitmap *pur_prd_bitmap,
                                                                    struct bitmap *next_pur_prd_bitmap,
                                                                    enum boolean_operator oper,
                                                                    struct bitmap *results_bitmap
                                                                    )
15 {
    unsigned int total_count = 0;

    total_count = (*pur_prd_select(query_type))(

                                                                    pur_prd_bitmap,
                                                                    next_pur_prd_bitmap,
                                                                    oper,
                                                                    results_bitmap );
20
    free_bitmap(pur_prd_bitmap);
    pur_prd_bitmap = NULL;
    free_bitmap(next_pur_prd_bitmap);
    next_pur_prd_bitmap = NULL;
    return(total_count);
}

25
/* Note: the _large indices deal with the original larger bitmap, the _reduced indices deal with the
smaller bitmap we are creating */
struct bitmap * reduce_bitmap ( struct bitmap * bitmap2, unsigned short *map, unsigned int number_of_items )
{
    struct bitmap * reduced_bitmap;
    unsigned short *sptr, map_count;
    unsigned int i, j, il_large, lk_large, il_reduced, lk_reduced, bit_large, bit_reduced;
30
    /* this is the resultant bitmap of size either pur or prd */
    if ((reduced_bitmap = create_general_bitmap ( number_of_items )) == NULL)
        error_handler (BITMAP_NOT_CREATE, ERROR, NO_STATUS, "reduced_bitmap in <reduce_bitmap>");
#ifdef DEBUG
    fprintf(debug_file, "%s\n", "REDUCE_BITMAP", "REDUCED_BITMAP", reduced_bitmap, "Mallocated");
    fprintf(debug_file, "%s\n", "REDUCE_BITMAP", "REDUCED_BITMAP->START", reduced_bitmap->start,
        "Mallocated");
    fprintf(debug_file, "%s\n", "REDUCE_BITMAP", "REDUCED_BITMAP->END", reduced_bitmap->end,
        "Mallocated");
#endif
35
    sptr = map;
    lk_large = lk_reduced = BITMAP_INTEGER_SIZE;
    il_large = il_reduced = 1;
    bit_large = bitmap2->start;
    bit_reduced = reduced_bitmap->start;
    for ( i=0; i < number_of_items; i++, sptr++ )
{

```



```

/* search map_count bits in the large bitmap corresponding to all the
product records for this purchase and if any bits are set, sets the bit
in the resultant bitmap, then continues bumping past rest of bits in
large bitmap. If no bits for this purchase are set, the proper bit in
the reduced bitmap is left off. */

5   map_count = *sptr;
   for (j=0; j<map_count; j++)
   {
       if ( *bit_large & j_large )
       {
           *bit_reduced |= j_reduced;
           break;
       }
       if ( *kk_large )
       {
10          j_large <<= 1;
       }
       else
       {
           bit_large++;
           kk_large = BITMAP_INTEGER_SIZE;
           j_large = 1;
       }
   }
   for (j=0; j<map_count; j++)
15  {
       if ( *kk_large )
       {
           j_large <<= 1;
       }
       else
       {
           bit_large++;
           kk_large = BITMAP_INTEGER_SIZE;
           j_large = 1;
20          }
       }
       if ( *kk_reduced )
       {
           j_reduced <<= 1;
       }
       else
       {
25          bit_reduced++;
           kk_reduced = BITMAP_INTEGER_SIZE;
           j_reduced = 1;
       }
   }

   return(reduced_bitmap);

struct bitmap * explode_pur_prd_bitmap ( struct bitmap * bitmap2, unsigned short *map, unsigned int number_of_items )
{
30   struct bitmap * explode_bitmap;
   unsigned short *sptr, map_count;
   unsigned int i, j, j_large, kk_large, j_exploded, kk_exploded, *bit_large, *bit_exploded,
   unsigned int num_bits,
   unsigned int total_count;

   total_count = count_set_bits(bitmap2);

   /* this is the resultant bitmap of size either pur or prd */
   if ((explode_bitmap = create_general_bitmap ( number_of_items )) == NULL)
       error_handler (BITMAP_NOT_CREATE, ERROR, NO_STATUS, "explode_bitmap in <explode_pur_prd
35   bitmap>");
   #ifdef DEBUG
       fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "EXPLODE_BITMAP", "EXPLODE_BITMAP", explode_bitmap, "Mallocated");
       fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "EXPLODE_BITMAP", "EXPLODE_BITMAP->START", explode_bitmap->start,
       "Mallocated");
       fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "EXPLODE_BITMAP", "EXPLODE_BITMAP->END", explode_bitmap->end,
       "Mallocated");
       sendf

```

```

num_bits = bitmap2->number_of_bits;
sptr = map;
kk_large = kk_exploded = BITMAP_INTEGER_SIZE;
ll_large = ll_exploded = 1;
br_large = bitmap2->start;
br_exploded = explode_bitmap->start;

5
for ( i=0; i < num_bits; i++, sptr++)
{
    /* search map_count bits in the large bitmap corresponding to all the
    product records for this purchase and if any bits are set, sets the bit
    in the resultant bitmap, then continues bumping past rest of bits in
    large bitmap. If no bits for this purchase are set, the proper bit in
    the exploded bitmap is left off. */

    map_count = *sptr;
    for ( j=0; j<map_count; j++)
    {
        if ( ~br_large & ll_large )
        {
            br_exploded |= ll_exploded;
        }
        if ( ~kk_exploded )
        {
            ll_exploded <= 1;
        }
        else
        {
            br_exploded++;
            kk_exploded = BITMAP_INTEGER_SIZE;
            ll_exploded = 1;
        }
    }

    /*
    for ( ; j<map_count; j++)
    {
        if ( ~kk_exploded )
        {
            ll_exploded <= 1;
        }
        else
        {
            br_exploded++
            kk_exploded = BITMAP_INTEGER_SIZE
            ll_exploded = 1;
        }
    }

    if ( ~kk_large )
    {
        ll_large <= 1;
    }
    else
    {
        br_large++
        kk_large = BITMAP_INTEGER_SIZE;
        ll_large = 1;
    }
}

total_count = count_set_bits(explode_bitmap);
return(explode_bitmap);

35

```

5

10

Appendix C for U.S. Patent Application

15 DATABASE LINK SYSTEM
 BACKGROUND OF THE INVENTION
 Filed October 22, 1993

Appendix C: Set of Routines to Handle Distributions

20 © 1993 FDC, Inc.
 All Rights Reserved

25

30

35

```

/*
**
** MODULE: DISTRIBUTION.C
**
5  ** MODULE DESCRIPTION:
**
**   Set of routines to handle distributions.
**
**
** AUTHORS:
**
**   Sushil Pillai   Digital Equipment Corporation
**
10  ** CREATION DATE: 20-September-1992
**
** DESIGN ISSUES:
**
**   Could add several return codes for some of the routines rather than
**   just SUCCESS or FAILURE
**
** PORTABILITY ISSUES:
**
15  **   None
**
** MODIFICATION HISTORY:
**
**   20-September-1992 - Original
**   04-October-1992   - to handle sequential lists.
**   07-October-1992   - to handle frequencies..
**   08-October-1992   - to handle crosstabs.
**   13-May-1993       - Chanes Malmkog
**                       - Replaced all fail_and_exit routines with error_handler() calls
20  **   30-Sep-1993    - Chanes Malmkog
**                       - Update error_handler() calls to include ERROR message type
**
**/

/*
**
** INCLUDE FILES
**
25  **/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <file.h>
#include <descrip.h>
#include <errno.h>

30  #include "tdc_prototype"
#include "tdc_file_defn"
#include "tdc_dmq_types"
#include "tdc_macro_defn"
#include "tdc_error_numbers.h"

#define NUMBER_OF_CROSSTABS 2
#define LOW_VALUE 0
#define HIGH_VALUE 99999
#define LESS_THAN_VALUE -100
35  #define GREATER_THAN_VALUE -200

/* UNDERLINE has 32 - dash characters */
#define UNDERLINE "-----"

```

```

char      *get_string();

/* Lookup table generation value */
int lookup_count=0;
5  int create_table = FALSE;
  int create_x_table = FALSE;
  int create_y_table = FALSE;
  int create_z_table = FALSE;

  int max_row_value_count = 0;
  int max_col_value_count = 0;
  int max_page_value_count = 0;

10  int x_axis_length = 0;
  int y_axis_length = 0;
  int z_axis_length = 0;

/* DMO specific external declarations */
extern int      DMO_CONNECTED;
extern int      msg_offset;
extern char     *msg_put_buffer[MAX_BUFFERS];
extern char     msg_file_buffer[FILE_BUFFER_LENGTH];
extern short    pams_put_msg_type;

15  /* External declarations */
  extern struct bitmap      *subsidiary_bitmap[SUB_FILE_MAX];
  extern struct bitmap      *pur01_bitmap;
  extern unsigned short     *pur01_map;
  extern unsigned int        pur01_map_count;

  extern struct bitmap      *prd01_bitmap;
  extern unsigned short     *prd01_map;
  extern unsigned int        prd01_map_count;

20  extern struct bitmap      *universe_bitmap;
  extern struct bitmap      *master_bitmap;

  extern struct bitmap      *super_master_bitmap;

  extern int      purchase_query;
  extern int      product_query;
  extern struct bitmap      *master_purchase_bitmap;
  extern struct bitmap      *master_product_bitmap;

25  extern char     *query_keycode;
  extern char     *query_label;
  extern char     query_filename[16];

  extern struct fixed_string_type *fixed_field[NUM_FIXED_STRINGS];
  extern int      max_number_of_brs;
  extern int      br_counts[256];

30  /* Count of a customer bitmap */
  extern unsigned int      master_count;

/* for purchase or product queries, total count can be greater than master_count */
/* however, for subsidiaries queries, the total count can be smaller than the */
/* master count.
/* Example. A distribution for purchase field, with a customer query, will result */
/* in total values in the distribution being more than the master count. */
/* Or for a subsidiary field, with a customer query, can result in the */
/* total values in the distribution less than the master count. Hence it */
35  /* is desirable to use a distribution count to capture that value. */
  unsigned int      distribution_total_count=0;

  extern unsigned int      reset_segmentation;

```

```

/* for debugging purposes */
extern FILE *debug_file;

/* Variable declarations for NONE case in X2, X3 distribution */
5 int
  int current_pos=0;
  int max_num_values=1;
  int max_count=0;
  int dist_bucket_count=0;

enum creation_type { create, insert };

struct lookup_values
{
10   int lookup_idx;
  char *lookup_string;
  struct lookup_values *next;
};

struct lookup_values *lookup_buffer, *lookup_val_tail;

/* Variable declarations */
struct lookup_table_list_type *lookup_table=NULL;
struct lookup_table_type *lookup_table_head=NULL;
15 struct lookup_table_type *lookup_table_tail=NULL;
struct lookup_table_type *table_tail=NULL;
struct lookup_table_type *temp_tail=NULL;

/* This module contains the following routines:
struct bucket_node *determine_value_distribution(data, field_type, field_length, f_offset, value_bitmap);
int display_field_distribution(field);
struct value_node *show_value_distribution(data, field_type, field_length, field_offset, value_bitmap);
load_lookup_data(root_node, tail_node, field_number, buffer, field_type, min_freq, max_freq);
20 load_lookup_information(root_node, tail_node, field_number, table_name, field_type, min_freq, max_freq);
char *get_bucket_value(search_value, lookup_bucket_table, field_type);
struct list_value_node *show_sample_distribution(data, root_node, tail_node, field_name, field_number,
  field_type, field_length, lookup_sample_table, value_bitmap);
int display_sample_distribution(file_list, min_freq, max_freq);
struct lookup_table_list_type *load_sample_count_sampler(root_node, root_z_node, tail_node, table_
load_sample_count_data(root_node, root_z_node, tail_node, buffer, field_type, offset, x_
  axis_count, y_axis_count, z_axis_count);
name, field_type, offset, x_axis_count, y_axis_count, z_axis_count);
int get_bucket_number(search_value, lookup_bucket_table, field_type);
struct distribution_type *show_sample_count_distribution(data, root_node, tail_node, field_type, field_
length, num_fields, lookup_table, value_bitmap);
void free_distribution_type(distr_var);
void free_lookup_table_list_type(lookup_dist);
30 free_lookup_table_type(lookup_ptr);
int display_sample_cnt_distribution(file_list);
int show_field_distribution(field);
struct list_value_node *show_list_value_distribution(data, root_node, tail_node, field_number,
  field_type, field_length, f_offset, value_bitmap);
int show_list_field_distribution();
int write_field_distribution();

*/
35

```

```

/*
  Not being invoked by the windows program.
  This routine is called by display_field_distribution. It produces a distribution of a single field.
  This distribution is built using a tree and hence the results come sorted.
  Not current being used by the windows application.
5  */
struct bucket_node *determine_value_distribution(data, field_type, field_length, f_offset, value_bitmap)
struct dataset      *data;
enum data_type      field_type;
unsigned int         field_length;
int                  f_offset;
struct bitmap        *value_bitmap;
{
10     unsigned int  i, j, jj, kk;
    unsigned long   *long_item;
    int              *large_neg_integer_item;
    unsigned int     *large_integer_item;
    unsigned short   *medium_integer_item;
    unsigned char     *small_integer_item;
    unsigned int      *bit_item;
    unsigned int      *temp_counter;
    float             *floating_item;
    double            *double_item;
15     char            *string_value;
    char              *string_ptr;
    unsigned int      num_bits;

    struct bucket_node *root_node = NULL;

    string_value = malloc(field_length+1);
    CHECK_ALLOCATION(string_value, "string_value:determine_value_distribution()");

    temp_counter = value_bitmap->start;

20     /* num_bits points to the actual number of items we are dealing with */
    num_bits = data->number_of_items;

    switch (field_type)
    {
        case dollars
        case floating_point
        case large_integer
25         /* processes the value for large_integer_item */
        PROCESS_BIT_TABLE_PART_1( large_integer_item, unsigned int * );
            insert_bucket( large_integer_item, &root_node, field_type, field_length);
        /* sets the pointer for the next iteration */
        PROCESS_BIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;

        case negative_float
        /* processes the value for large_neg_integer_item */
        PROCESS_BIT_TABLE_PART_1( large_neg_integer_item, int * );
            insert_bucket( large_neg_integer_item, &root_node, field_type, field_length);
30         /* sets the pointer for the next iteration */
        PROCESS_BIT_TABLE_PART_2( large_neg_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;

        case year_month
        case year_month_day
        case medium_integer
        PROCESS_BIT_TABLE_PART_1( medium_integer_item, unsigned short * );
            insert_bucket( medium_integer_item, &root_node, field_type, field_length);
        PROCESS_BIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
35     case character
        case small_integer
    }

```

```

PROCESS_BIT_TABLE_PART_1( small_integer_item, unsigned char * );
    insert_bucket( "small_integer_item", &root_node, field_type, field_length );
PROCESS_BIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
break;
5 case string_type
    PROCESS_BIT_TABLE_PART_1( string_ptr, unsigned char * );
        strcpy( string_value, string_ptr, field_length );
        string_value[field_length] = '\0';
        insert_bucket( string_value, &root_node, field_type, field_length );
    PROCESS_BIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE, field_length, field_length );
    break;
case bit_type
    PROCESS_BIT_TABLE_PART_1( bit_item, unsigned int * );
        insert_bucket( ("bit_item&jj"), &root_node, field_type, field_length );
10 PROCESS_BIT_TABLE_PART_2( bit_item, BITMAP_INTEGER_SIZE, 1 );
    break;
case fixed_string:
    PROCESS_BIT_TABLE_PART_1( medium_integer_item, unsigned short * );
        strcpy( string_value, fixed_field[fixed_offset] -> fixed_string( "medium_integer_item" ) ->
            string, field_length );
        string_value[field_length] = '\0';
        insert_bucket( string_value, &root_node, field_type, field_length );
15 PROCESS_BIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
    break;

}
free( string_value );
string_value = NULL;

return( root_node );

20 }
/*
Not being invoked by the windows program
Called by the parser, using the "DP" code
It provides a distribution for a single field. This functionality is not being used anymore
*/
int display_field_distribution( field )
struct field_entry field;
{
25
    int status;
    int char;
    struct address_range reladdr;
    struct bucket_node *distribution = NULL;
    char *buffer;
    char msg_buffer[132];
    int i, field_offset=0;

    if (master_count == 0)
30 {
        strcpy( msg_buffer, "Query did not produce any results.\n");
        strcat( msg_buffer, "No distribution created.\n\n");
        if (DMQ_CONNECTED)
            msg_buf_xfer( msg_buffer );
        else
            printf( "%s", msg_buffer );

        return( SUCCESS );
35 }

```



```

/* map section file */
status = OpenMapFile(field->table, "Toobar.sec", field->field_name, &chan,
                    &retadr, field->von, field->number_of_blocks);
if (is_error(status))
5   error_handler (MAP_OPEN_ERR, ERROR, status, "display_field_distribution");

/* set start address of data points */
field->data->items = retadr.start;

if (field->field_type == fixed_string)
    field_offset = get_fixed_string_offset(field->field_name);

/* search for value distribution */
10  distribution = determine_value_distribution(field->data, field->field_type,
                                             field->field_end, field_offset, master_bmap);

/* print value distribution */
buffer = malloc(STR_BUFFER_LENGTH);
CHECK_ALLOCATION(buffer, "buffer.Routine display_field_distribution()");
sprintf(buffer, "Value distribution for %s:\n", field->field_name);
msg_buf_xfer ( buffer );

free( buffer );
15  buffer = NULL;

if (distribution != NULL)
{
    /* print_bucket_tree will free the value nodes as prints are completed */
    print_bucket_tree(distribution, field->field_type,
                     field->field_precision, field->field_end);
}
else
{
20  sprintf(buffer, "No values found for this distribution, please try another query.\n");
    msg_buf_xfer ( buffer );
    print_text_bufs( );
}

/* unmap section file */
status = UnmapCloseFile(chan, &retadr);

if (is_error(status))
25  error_handler (UNMAP_CLOSE_ERR, ERROR, status, "display_field_distribution");

return(SUCCESS);
}

/*
This routine builds the linked list based on the table entries.
*/
load_lookup_data(root_node, tail_node, field_number, buffer, field_type, field_length, min_freq, max_freq)
30  struct list_value_node *root_node;
    struct value_node *tail_node;
    unsigned int field_number;
    char *buffer;
    enum data_type field_type;
    int field_length;
    int min_freq;
    int max_freq;
{
    struct lookup_table_type *temp_table = NULL;
    char *temp_char;
    char *mod_buffer = NULL;
    char *delimiter=LOOKUP_DELIMITER;
    char low_date[DATE_STRING_LENGTH], high_date[DATE_STRING_LENGTH];
    char high_char, low_char;
    int temp_length;
35

```

```

temp_table = (struct lookup_table_type *) malloc (sizeof(*temp_table));
CHECK_ALLOCATION(temp_table, "Routine load_lookup_data: temp_table");
#ifdef DBG
fprintf(debug_file, "%s\n", "LOAD_LOOKUP_DATA", "TEMP_TABLE",
temp_table, "Mallocated and
initially used by lookup_table_head and lookup_table_tail");
5 #endif
temp_table->next = NULL;

mod_buffer = buffer;
switch(field_type)
{
    case character:
        temp_table->value = get_string(&mod_buffer, delimiter);
10 temp_char = get_string(&mod_buffer, delimiter);
temp_table->low = temp_char;
free(temp_char);
temp_char = get_string(&mod_buffer, delimiter);
temp_table->high = temp_char;
free(temp_char);
temp_char = NULL;
temp_table->total=0;
15 break;
    case bit_type:
    case small_integer:
    case fixed_string:
    case medium_integer:
    case large_integer:
    case dollars:
    case negative_float:
    case floating_point:
20 temp_table->value = get_string(&mod_buffer, delimiter);
temp_char = get_string(&mod_buffer, delimiter);
temp_table->low = atoi(temp_char);
free(temp_char);
temp_char = get_string(&mod_buffer, delimiter);
temp_table->high = atoi(temp_char);
free(temp_char);
temp_char = NULL;
25 temp_table->total=0;
break;
    case year_month:
    case year_month_day:
temp_table->value = get_string(&mod_buffer, delimiter);
temp_char = get_string(&mod_buffer, delimiter);
temp_table->low = date_to_number(temp_char);
free(temp_char);
30 temp_char = get_string(&mod_buffer, delimiter);
temp_table->high = date_to_number(temp_char);
free(temp_char);
temp_char = NULL;
temp_table->total=0;
break;
    case string_type:
35 temp_table->value = get_string(&mod_buffer, delimiter);
temp_char = get_string(&mod_buffer, delimiter);
temp_table->low = atoi(temp_char);
free(temp_char);
temp_char = get_string(&mod_buffer, delimiter);
temp_table->high = atoi(temp_char);
free(temp_char);
temp_char = NULL;
temp_table->total=0;
break;

```

```

    case bad_data_type.
        break;
    }

    /* determine the length of the field value */
    temp_length = strlen(temp_table->value);
    if (x_axis_length < temp_length)
        x_axis_length = temp_length;
5   if (lookup_table_head == NULL)
    {
        lookup_table_head = temp_table;
        lookup_table_tail = temp_table;
    }
    else
    {
        lookup_table_tail->next = temp_table;
        lookup_table_tail = temp_table;
    }
10  }

    /*
    This routine builds the lookup table (linked list). For a distribution, a table name (filename) or NONE can be
    specified. If a table name is specified the entries of that file are build as linked list. However, for the NONE
    case, we do nothing, but handle the build of the table dynamically during get_bucket_value.
    */
    load_lookup_information(root_node, tail_node, value, field_number, table_name, field_name,
                           field_type, field_length, min_freq, max_freq, reference_count)

15  struct list_value_node  *root_node;
    struct value_node       *tail_node;
    void                    *value;
    unsigned int            field_number;
    char                    table_name;
    char                    field_name;
    enum data_type          field_type;
    int                     field_length;
    int                     min_freq;
    int                     max_freq;
    int                     reference_count;
20  {
    FILE                    *f;
    struct lookup_table_type *temp_table;
    char                    buffer[MAXCHARS];
    char                    *mod_buffer=NULL, *temp_string=NULL, *file_ptr=NULL;
    int                     i, low, high, projection, difference=10;
    char                    *temp_table_name;
    enum pur_pro_type       query_mode;
    enum pur_pro_query_type query_type;
25  mod_buffer = table_name;

    if (table_name == NULL)
        return(0);
    else
    {
        temp_table_name = strdup(table_name);
        if (strcmp(temp_table_name, "LOOKUP_DIR NONE DAT.", 20) == 0)
30  {
            create_table = TRUE;
            max_count = MAX_DIST_COUNT;
            return(1);
        }
        free(temp_table_name);
        temp_table_name = NULL;
    }
}

    /*
    A table name will only have a '-', if instead of a table name a range was specified. In which case,
    I dynamically create the list of values. However, this functionality is not being used.
35

```

```

*/
if ((file_ptr = strchr(table_name, '.')) != NULL)
{
    temp_string = get_string(&mod_buffer, '.');
    sscanf(temp_string, "%d", &low);
    free(temp_string);
    temp_string = NULL;
5   sscanf(mod_buffer, "%d", &high);
    if ((int)(high/difference) <= 1)
        difference = 1;
    if ((projection = (int)(high/difference)) <= 0)
        projection++;
    for (i = low; i <= projection; i++)
    {
        sprintf(buffer, "%d+.%d.%d", low, low, (low+difference));
        load_lookup_data(root_node.tail_node.field_number, buffer,
10         field_type, field_length, min_freq, max_freq);
        low += (difference+1);
    }
}
else
{
    /* If a table name was specified */
    if ((f = fopen(table_name, "r")) != NULL)
    {
        while (fgetc(buffer, MAXCHARS, f) != NULL)
15         {
            load_lookup_data(root_node.tail_node.field_number, buffer, field_type,
                                field_length, min_freq, max_freq);
        }
    }
    else
    {
        error_handler(FILE_NOT_OPEN, ERROR, NO_STATUS, table_name);
    }
    if (fclose(f) == EOF)
20     error_handler(FILE_NOT_CLOSE, ERROR, NO_STATUS, table_name);
}

/* the OTHER case */
if (x_axis_length < 5)
    x_axis_length = 5;

/* for the OTHER case */
temp_table = (struct lookup_table_type *) malloc(sizeof(*temp_table))
25 CHECK_ALLOCATION(temp_table, "Routine load_lookup_information temp_table")
#ifdef DBG
fprintf(debug_file, "%s\n", "LOAD_LOOKUP_INFORMATION", "TEMP_TABLE", temp_table, "Mallocated");
#endif
temp_table->next = NULL;

temp_table->value = malloc(FIELD_NAME_LENGTH+1)
CHECK_ALLOCATION(temp_table->value, "temp_table->value, load_lookup_information:temp_table->value");
#ifdef DBG
30 fprintf(debug_file, "%s\n", "LOAD_LOOKUP_INFORMATION", "TEMP_TABLE->VALUE", temp_table->value,
    "Mallocated");
#endif
strcpy(temp_table->value, "OTHER", 5); /* 5 is length of text OTHER */
temp_table->value[5] = '\0';
temp_table->low = LOW_VALUE;
temp_table->high = HIGH_VALUE;
temp_table->low1 = 0;

/* append OTHER to the list */
35 lookup_table_tail->next = temp_table;
lookup_table_tail = temp_table;
}

```

```

/*
  When a value needs to be inserted either at the beginning or end, this routine will initially create the structure and then
  do the appropriate insertion.
*/
create_lookup_table(mode, root_node, tail_node, field_name, field_number, search_value, field_type, field_precision, tail)
enum creation_type      mode;
struct list_value_node   *root_node;
struct value_node        *tail_node;
char                    *field_name;
5 unsigned int           field_number;
void                    *search_value;
enum data_type           field_type;
int                     field_precision;
struct lookup_table_type *tail;
{
    struct lookup_table_type *temp_table = NULL, *temp_ptr;
    char                    string_val[FIELD_NAME_LENGTH];
    int                     field_offset;
10    int                 med_integer;
    float                 float_num;
    int                     len;
    int                     prec_val;
    int                     temp_length;

    memset(string_val, 0, FIELD_NAME_LENGTH);
    temp_table = (struct lookup_table_type *) malloc (sizeof(*temp_table));
    CHECK_ALLOCATION(temp_table, "Routine create_lookup_table: temp_table");
    #ifdef DBG
15    fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE", temp_table, "Mallocated");
    #endif
    temp_table->next = NULL;

    switch(field_type)
    {
        case small_integer:
            temp_table->low = temp_table->high = (unsigned char *) search_value;
            temp_table->total = 1;
            sprintf(string_val, "%d", (unsigned char *) search_value);
            temp_table->value = malloc(FIELD_NAME_LENGTH+1);
            CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
20            table->value");
            #ifdef DBG
            fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
            "Mallocated");
            #endif
            strcpy(temp_table->value, string_val);
            break;
25        case character:
            temp_table->low = temp_table->high = (unsigned char *) search_value;
            temp_table->total = 1;
            sprintf(string_val, "%c", (unsigned char *) search_value);
            temp_table->value = malloc(FIELD_NAME_LENGTH+1);
            CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
            table->value");
            #ifdef DBG
            fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
            "Mallocated");
30            #endif
            strcpy(temp_table->value, string_val);
            break;
        case year_month:
        case year_month_day:
            temp_table->low = temp_table->high = (unsigned short *) search_value;
            temp_table->total = 1;
            temp_table->value = malloc(FIELD_NAME_LENGTH+1);
            CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp
35            table->value");
            #ifdef DBG
            fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
            "Mallocated");
            #endif
            strcpy(temp_table->value, number_to_date((unsigned short *) search_value));
            break;
        case fixed_string:
            med_integer = temp_table->low = temp_table->high = (unsigned short *) search_value;
            temp_table->total = 1;
    }
}

```

128

```

    field_offset = get_fixed_string_offset(field_name);
    if (fixed_field[field_offset] -> fixed_string(integer) != NULL)
        strcpy(string_val, fixed_field[field_offset] -> fixed_string(integer) -> string);
    else
        sprintf(string_val, "%d", (unsigned short *) search_value);

    temp_table->value = malloc(FIELD_NAME_LENGTH+1);
    CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
5  #ifdef DBG
    fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
        "Malloced.");
    #endif

    strcpy(temp_table->value, string_val);
    break;
    case medium_integer:
        temp_table->low = temp_table->high = (unsigned short *) search_value;
        temp_table->total = 1;
        sprintf(string_val, "%d", (unsigned short *) search_value);
        temp_table->value = malloc(FIELD_NAME_LENGTH+1);
        CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
10 #ifdef DBG
        fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
            "Malloced.");
        #endif

        strcpy(temp_table->value, string_val);
        break;
        case dollars:
            temp_table->low = temp_table->high = (unsigned int *) search_value;
            temp_table->total = 1;
            float_num = (float)temp_table->low/100.0;
            sprintf(string_val, "%7.2f", float_num);
            temp_table->value = malloc(FIELD_NAME_LENGTH+1);
            CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
15 #ifdef DBG
            fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
                "Malloced.");
            #endif

            strcpy(temp_table->value, string_val);
            break;
            case floating_point:
                prec_val = (int)pow(10, field_precision);
                temp_table->low = temp_table->high = (unsigned int *) search_value;
                temp_table->total = 1;
                float_num = (float)temp_table->low/prec_val;
                sprintf(string_val, "%7.4f", field_precision, float_num);
                temp_table->value = malloc(FIELD_NAME_LENGTH+1);
                CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
20 #ifdef DBG
                fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
                    "Malloced.");
                #endif

                strcpy(temp_table->value, string_val);
                break;
                case negative_float:
                    prec_val = (int)pow(10, field_precision);
                    temp_table->low = temp_table->high = (int *) search_value;
                    temp_table->total = 1;
                    float_num = (float)temp_table->low/prec_val;
                    sprintf(string_val, "%7.4f", field_precision, float_num);
                    temp_table->value = malloc(FIELD_NAME_LENGTH+1);
                    CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
25 #ifdef DBG
                    fprintf(debug_file, "%s\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
                        "Malloced.");
                    #endif

                    strcpy(temp_table->value, string_val);
                    break;
                    case large_integer:
                    case bit_type:
                        temp_table->low = temp_table->high = (unsigned int *) search_value;
                        temp_table->total = 1;
                        sprintf(string_val, "%d", (unsigned int *) search_value);
30
35

```

129

```

temp_table->value = malloc(FIELD_NAME_LENGTH+1);
CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
table->value");
#ifdef DBG
fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
"Malloced.");
#endif

5      strcpy(temp_table->value, string_val);
      break;
      case string_type
      sprintf(string_val, "%s", (unsigned char *) search_value);
      temp_table->low = temp_table->high = 0;
      temp_table->total = 1;
      temp_table->value = malloc(FIELD_NAME_LENGTH+1);
      CHECK_ALLOCATION(temp_table->value, "Routine create_lookup_table: temp_
table->value");
#ifdef DBG
10     fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "CREATE_LOOKUP_TABLE", "TEMP_TABLE->VALUE", temp_table->value,
"Malloced.");
#endif

      strcpy(temp_table->value, string_val);
      break;
      case bad_data_type
      break;
}

/* determine the length of the field value */
temp_length = strlen(temp_table->value);
15 if (x_axis_length < temp_length)
    x_axis_length = temp_length;

if (lookup_table_head == NULL)
{
    lookup_table_head = temp_table;
    lookup_table_tail = temp_table;
}
else
20 {
    if (mode == insert)
    {
        if (tail == lookup_table_head)
        {
            if ((field_type == string_type) && (strcmp(temp_table->value, table->value) < 0))
            {
                temp_table->next = lookup_table_head;
                lookup_table_head = temp_table;
            }
            else
            {
                if (temp_table->low < tail->low)
                {
                    temp_table->next = lookup_table_head;
                    lookup_table_head = temp_table;
                }
                else
                {
                    temp_ptr = tail->next;
                    tail->next = temp_table;
                    temp_table->next = temp_ptr;
                    if (tail == lookup_table_tail)
                    {
                        lookup_table_tail = temp_table;
                    }
                }
            }
        }
        else
        {
            temp_ptr = tail->next;
            tail->next = temp_table;
            temp_table->next = temp_ptr;
            if (tail == lookup_table_tail)
            {
                lookup_table_tail = temp_table;
            }
        }
    }
}
else
35 {
    temp_ptr = tail->next;
    tail->next = temp_table;
    temp_table->next = temp_ptr;
    if (tail == lookup_table_tail)
    {
        lookup_table_tail = temp_table;
    }
}
}

```

```

    }
    else
    {
        lookup_table_tail->next = temp_table;
        lookup_table_tail = temp_table;
    }
}

```

10 Every value read from the database is compared to the table of values (linked list). If a value is found the total counter is incremented. If the value is not found but less than the first table entry, the new value is inserted before the list. If the value is not in the existing list, the new value is inserted after the list.

After a pre-determined of max_count, values not found in the list are inserted in the OTHER bucket, irrespective of if they were less than the first value or greater than all values.

max_count in this case is equal to MAX_Y_AXIS_COUNT, which is current set to 100 (i.e. only 100 values will be displayed).

```

15 int get_bucket_value(root_node, tail_node, field_name, field_number, search_value, field_type, field_precision, field_length)
    struct list_value_node *root_node;
    struct value_node *tail_node;
    char *field_name;
    unsigned int field_number;
    void *search_value;
    enum data_type field_type;
    int field_precision;
    int field_length;
    {
        struct lookup_table_type *head, *tail, *temp_table=NULL;
        int status;
        enum creation_type mode;
        int i;
        int string_value_num;

        temp_table = lookup_table_head;
        tail = temp_table;

        if ((temp_table == NULL) && (create_table))
        {
            /* NONE case */
            mode = create;
            create_lookup_table(mode, root_node, tail_node, field_name, field_number, search_value, field_type, field_
            precision, NULL); dist_bucket_count++;
            return(1);
        }

        status = 0;
        for (; temp_table != NULL; temp_table = temp_table->next)
        {
            switch(field_type)
            {
                case dollars:
                case floating_point:
                case large_integer:
                case bit_type:
                    if ((unsigned int *) search_value < temp_table->low)
                    {
                        status = LESS_THAN_VALUE;
                        break;
                    }
                    if ((unsigned int *) search_value <= temp_table->high)
                    {
                        temp_table->total++;
                        return(1);
                    }
            }
        }
    }

```



```

        }
        break;
    case negative_float:
        if ((int *) search_value < temp_table->low)
        {
            status = LESS_THAN_VALUE;
            break;
        }
        if ((int *) search_value <= temp_table->high)
        {
            temp_table->total++;
            return( 1 );
        }
        break;
    case small_integer:
    case character:
        if ((unsigned char *) search_value < temp_table->low)
        {
            status = LESS_THAN_VALUE;
            break;
        }
        if ((unsigned char *) search_value <= temp_table->high)
        {
            temp_table->total++;
            return( 1 );
        }
        break;
    case fixed_string:
    case medium_integer:
    case year_month:
    case year_month_day:
        if ((unsigned short *) search_value < temp_table->low)
        {
            status = LESS_THAN_VALUE;
            break;
        }
        if ((unsigned short *) search_value <= temp_table->high)
        {
            temp_table->total++;
            return( 1 );
        }
        break;
    case string_type:
        if ( ( t = strcmp( search_value, temp_table->value, field_length) ) < 0 )
        {
            status = LESS_THAN_VALUE;
            break;
        }
        if ( t )
        {
            temp_table->total++;
            return( 1 );
        }
        break;
    }

    if ( status )
        break;

    tail = temp_table;
}

if (create_table)
{
    /* table is NONE */
    if (dist_bucket_count < max_count)
    {

```

```

mode = insert;
create_lookup_table(mode, root_node, tail_node, field_name, field_number, search_value, field_type,
field_precision, tail); dist_bucket_count++;
return(1);
}
else
5 {
    if (dist_bucket_count == max_count)
    {
        /* for the OTHER case */
        head = (struct lookup_table_type *) malloc (sizeof(*head));
        CHECK_ALLOCATION(head, "Routine get_bucket_value: head");
        #ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_BUCKET_VALUE", "HEAD", head, "Mallocated.");
        #endif
10        head->next = NULL;

        head->value = malloc(FIELD_NAME_LENGTH+1);
        CHECK_ALLOCATION(head->value, "head->value: get_bucket_value: head->value");
        #ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_BUCKET_VALUE", "HEAD->VALUE", head->value, "Mallocated.");
        #endif
15        strcpy(head->value, "OTHER", 5); /* 5 is length of text OTHER */
        head->value[5] = '\0';
        head->low = LOW_VALUE;
        head->high = HIGH_VALUE;
        head->total = 1;

        lookup_table_tail->next = head;
        lookup_table_tail = head;
        dist_bucket_count++;
        return(1);
    }
    else
20 {
        lookup_table_tail->total++;
        return(1);
    }
}
else /* not creating tables, using table this should be using OTHER table */
{
    lookup_table_tail->total++;
    return(1);
}
25 }

/*
Actually builds the DT distribution. This routine will be called either for a customer distribution, or for a
purchase/product distribution ONLY IF a purchase/product query was found
*/
struct list_value_node *show_sample_distribution(data, root_node, tail_node, field_name,
field_number, field_type, field_precision, field_length, value_bitmap)
30 struct dataset
{
    struct list_value_node *root_node;
    struct list_value_node *tail_node;
    char *field_name;
    unsigned int field_number;
    enum data_type field_type;
    int field_precision;
    unsigned int field_length;
    struct bitmap *value_bitmap;
}
35

```

```

unsigned int i, j, k;
unsigned int num_bits;
unsigned long *long_item;
int *large_neg_integer_item;
5 unsigned int *large_integer_item;
unsigned short *medium_integer_item;
unsigned char *small_integer_item;
unsigned int *bit_item;
unsigned int *temp_counter;
float *floating_item;
double *double_item;
char *string_value;
char *string_ptr;

10 string_value = malloc(field_length+1);
CHECK_ALLOCATION(string_value, "string_value:show_sample_distribution()");
#ifdef DBG
printf(debug_file, "%s\n", "SHOW_SAMPLE_DISTRIBUTION", "STRING_VALUE", string_value, "Mallocated");
#endif

temp_counter = value_bitmap->start;

num_bits = data->number_of_items;

15 switch (field_type)
{
    case dollars:
    case floating_point:
    case large_integer:
        PROCESS_BIT_TABLE_PART_1( large_integer_item, unsigned int * );
        get_bucket_value(root_node, tail_node, field_name, field_number, *large_integer_item,
                        field_type, field_precision, field_length);
        PROCESS_BIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
20 case negative_float:
        PROCESS_BIT_TABLE_PART_1( large_neg_integer_item, int * );
        get_bucket_value(root_node, tail_node, field_name, field_number, *large_neg_integer_item,
                        field_type, field_precision, field_length);
        PROCESS_BIT_TABLE_PART_2( large_neg_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case fixed_string:
    case year_month:
    case year_month_day:
25 case medium_integer:
        PROCESS_BIT_TABLE_PART_1( medium_integer_item, unsigned short * );
        get_bucket_value(root_node, tail_node, field_name, field_number, *medium_integer_item,
                        field_type, field_precision, field_length);
        PROCESS_BIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case character:
    case small_integer:
30 PROCESS_BIT_TABLE_PART_1( small_integer_item, unsigned char * );
        get_bucket_value(root_node, tail_node, field_name, field_number, *small_integer_item,
                        field_type, field_precision, field_length);
        PROCESS_BIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case string_type:
        PROCESS_BIT_TABLE_PART_1( string_ptr, unsigned char * );
        strcpy(string_value, string_ptr, field_length);
        string_value[field_length] = '\0';
        get_bucket_value(root_node, tail_node, field_name, field_number, string_value,
                        field_type, field_precision, field_length);
35 PROCESS_BIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE*field_length, field_length );
        break;
    case bit_type:
        PROCESS_BIT_TABLE_PART_1( bit_item, unsigned int * );
        get_bucket_value(root_node, tail_node, field_name, field_number, (*bit_item&j),
                        field_type, field_precision, field_length);
        PROCESS_BIT_TABLE_PART_2( bit_item, BITMAP_INTEGER_SIZE, 1 );
        break;

```

134

```

        field_type, field_precision, field_length);
        break;
    case fixed_string:
    case year_month:
    case year_month_day:
    case medium_integer:
        PROCESS_MBIT_TABLE_PART_1( medium_integer_item, unsigned short * );
        get_bucket_value( root_node, tail_node, field_name, field_number, "medium_integer_item",
            field_type, field_precision, field_length);
        PROCESS_MBIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case character:
    case small_integer:

    )
    #ifdef DBG
        fprintf(debug_file, "%s\n", "SHOW_SAMPLE_DISTRIBUTION", "STRING_VALUE", string_value, "Freed ");
    #endif
5   free( string_value );
    string_value = NULL;

    return( root_node );
}

/*
    Actually builds the distribution. This routine will be called for a subsidiary distribution.
*/
10 struct list_value_node *show_subs_sample_distribution( data, root_node, tail_node, field_name,
    field_number, field_type, field_precision, field_length, value_bitmap, reference_bitmap )
struct dataset
    struct list_value_node *root_node;
    struct value_node *tail_node;
    char *field_name;
    unsigned int field_number;
    enum data_type field_type;
15 int field_precision;
    unsigned int field_length;
    struct bitmap *value_bitmap;
    struct bitmap *reference_bitmap;
    {
        unsigned int i, j, jj, kk, ll, mm;
        unsigned int num_bits;
        unsigned long *long_item;
20 int *large_neg_integer_item;
        unsigned int *large_integer_item;
        unsigned short *medium_integer_item;
        unsigned char *small_integer_item;
        unsigned int *bit_item;
        unsigned int *temp_counter, *temp_reference;
        float *floating_item;
        double *double_item;
25 char *string_value;
        char *string_ptr;

        temp_counter = value_bitmap->start;
        temp_reference = reference_bitmap->start;

        num_bits = value_bitmap->number_of_bits;
30 switch (field_type)

```

```

case dollars:
case floating_point:
case large_integer:
    PROCESS_MBIT_TABLE_PART_1( large_integer_item, unsigned int * );
    get_bucket_value( root_node, tail_node, field_name, field_number, *large_integer_item,
                      field_type, field_precision, field_length );
35    PROCESS_MBIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
    break;
case negative_float:
    PROCESS_MBIT_TABLE_PART_1( large_neg_integer_item, int * );
    get_bucket_value( root_node, tail_node, field_name, field_number, *large_neg_integer_item,
                      field_type, field_precision, field_length );

    PROCESS_MBIT_TABLE_PART_1( small_integer_item, unsigned char * );
    get_bucket_value( root_node, tail_node, field_name, field_number, *small_integer_item,
                      field_type, field_precision, field_length );
    PROCESS_MBIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
    break;
case string_type:
    string_value = malloc( field_length + 1 );
    CHECK_ALLOCATION( string_value, "string_value, show subs sample" );
5    #ifdef DBG
    fprintf( debug_file, "%s\n", "SHOW_SUBS_SAMPLE_DISTRIBUTION", "STRING_VALUE", string_value, "Malloced" );
    #endif

    PROCESS_MBIT_TABLE_PART_1( string_ptr, unsigned char * );
    strcpy( string_value, string_ptr, field_length );
    string_value[ field_length ] = '\0';
    get_bucket_value( root_node, tail_node, field_name, field_number, string_value,
                      field_type, field_precision, field_length );
10    PROCESS_MBIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE, field_length, field_length );
    #ifdef DBG
    fprintf( debug_file, "%s\n", "SHOW_SUBS_SAMPLE_DISTRIBUTION", "STRING_VALUE", string_value, "Freed" );
    #endif

    free( string_value );
    string_value = NULL;
    break;
case bit_type:
    PROCESS_MBIT_TABLE_PART_1( bit_item, unsigned int * );
    get_bucket_value( root_node, tail_node, field_name, field_number, (*bit_item & 1),
                      field_type, field_precision, field_length );
15    PROCESS_MBIT_TABLE_PART_2( bit_item, BITMAP_INTEGER_SIZE, 1 );
    break;
}

return( *root_node );
}

/*
20    Actually builds the distribution. This routine will be called for a purchase/product distribution. If a customer query
    was executed. Customer query is a query, which results in a customer bitmap
*/
struct list_value_node *show_pur_pro_sample_dist( data root_node, tail_node, field_name, field_number, field_type, field_precision,
field_length, value_bitmap, query_mode, query_type, pur_pro_count )
{
    struct dataset
    {
        struct list_value_node *root_node;
        struct value_node *tail_node;
        char *field_name;
        unsigned int field_number;
        enum data_type field_type;
        int field_precision;
        unsigned int field_length;
        struct bitmap *value_bitmap;
        enum pur_pro_type query_mode;
        enum pur_pro_query_type query_type;
        int *pur_pro_count;
    };
    struct dataset data;
    data.root_node = root_node;
    data.tail_node = tail_node;
    data.field_name = field_name;
    data.field_number = field_number;
    data.field_type = field_type;
    data.field_precision = field_precision;
    data.field_length = field_length;
    data.value_bitmap = value_bitmap;
    data.query_mode = query_mode;
    data.query_type = query_type;
    data.pur_pro_count = pur_pro_count;

    unsigned int i, j, kk, ll, m=0, mm;
    unsigned int num_bits, num_values=1;
    unsigned long *long_item;
    int *large_neg_integer_item;
}

```

```

    unsigned int *large_integer_item;
    unsigned short *medium_integer_item;
    unsigned char *small_integer_item;
    unsigned int *bit_item;
    unsigned int temp_counter;
    unsigned short temp_map;
    float *floating_item;
    5 double *double_item;
    char *string_value;
    char *string_ptr;
    temp_counter = value_bitmap->start;

    if (query_mode == purchase)
        temp_map = pur01_map;
    else
        if (query_mode == product)
            temp_map = prd01_map;

    num_bits = value_bitmap->number_of_bits;
    10 switch (field_type)
    {
    case dollars:
    case floating_point:
    case large_integer:
        PROCESS_PBIT_TABLE_PART_1( large_integer_item, unsigned int * ),
            (*pur_prd_count)++;
            get_bucket_value(root_node.tail_node.field_name field_number,
                (*large_integer_item+mm)), field_type field_precision field_length);
        15 PROCESS_PBIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case negative_float:
        PROCESS_PBIT_TABLE_PART_1( large_neg_integer_item, int * ),
            (*pur_prd_count)++;
            get_bucket_value(root_node.tail_node.field_name field_number,
                (*large_neg_integer_item+mm)), field_type field_precision field_length);
        PROCESS_PBIT_TABLE_PART_2( large_neg_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    20 case fixed_string
    case year_month
    case year_month_day:
    case medium_integer:
        PROCESS_PBIT_TABLE_PART_1( medium_integer_item, unsigned int * )
            (*pur_prd_count)++;
            get_bucket_value(root_node.tail_node.field_name field_number,
                (*medium_integer_item+mm)), field_type field_precision field_length);
        PROCESS_PBIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    25 case character:
    case small_integer:
        PROCESS_PBIT_TABLE_PART_1( small_integer_item, unsigned int * );
            (*pur_prd_count)++;
            get_bucket_value(root_node.tail_node.field_name field_number,
                (*small_integer_item+mm)), field_type field_precision field_length);
        PROCESS_PBIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case string_type
    30 string_value = malloc(field_length+1);
        CHECK_ALLOCATION(string_value, "string_value: show_pur_prd_sample_distribution()"); #ifdef DBG
        fprintf(debug_file, "%s\n", "SHOW_PUR_PRD_SAMPLE_DISTRIBUTION", "STRING_VALUE", string_value,
            "Malloced.");
        sendif

        PROCESS_PBIT_TABLE_PART_1( string_ptr, unsigned char * );
            (*pur_prd_count)++;
            /* djl - svp => if create_lookup_table had string length and we used new
            hash_xchar call in get_bucket_value, we could avoid strcpy here */
            35 strcpy(string_value, (string_ptr+(mm*field_length)), field_length);
            string_value[field_length] = '\0';
            get_bucket_value(root_node.tail_node.field_name field_number, string_value,
                field_type field_precision field_length);
        PROCESS_PBIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE field_length, field_length );
    
```

137

```

35 #ifdef DBG
    fprintf(debug_file, "%s\t%s\t%d\t%s\n", "SHOW_PUR_PRD_SAMPLE_DISTRIBUTION", "STRING_VALUE", string_value, "Freed.");

    #endif

    free ( string_value );
    string_value = NULL;
    break;

    case bit_type:
        PROCESS_PBIT_TABLE_PART_1( bit_rem, unsigned int * );
        ("pur_prd_count")++;
        get_bucket_value(root_node.tail_node.field_name.field_number,
            ("bit_rem+mm"), field_type, field_precision, field_length);
        PROCESS_PBIT_TABLE_PART_2( bit_rem, BITMAP_INTEGER_SIZE, 1 );
        break;
    }

    return(root_node);
}

/*
10 This is the "DT" case. It is being called by the parser as "display distribution". It provides a sorted
distribution of a single field, using linked list, as opposed to a tree.
*/
int display_sample_distribution(file_list, min_freq, max_freq)
char *file_list;
int min_freq;
int max_freq;
{
    int status;
    int chan;
    int i=0;
    char *mod_buffer=NULL;
    char *file_ptr;
    char table_name[MAXFIELDS];
    char filename=NULL;
    struct address_range reloc;
    struct query_info *tempq;
    struct field_entry *field;
    unsigned int field_number;
    int num_fields=0;
    char delimiter=LOOKUP_DELIMITER;
    struct list_value_node *distribution;
    struct list_value_node *root_node = NULL;
    struct value_node *tail_node = NULL;
    char *buffer, *field_buffer;
    char msg_buffer[132];
    int reference_count=0;
    int field_length;
    char *master_count_str, temp_count_str[NUM_STRING_SIZE];
    enum pur_prd_query_type query_type;
    struct query_info *query_tree_head = NULL;
    enum pur_prd_type query_mode;
    unsigned int pur_prd_count=0;

    /* initialize global values */
    x_axis_length = 0;
    distribution_total_count = 0;
    max_count = dist_bucket_count = 0;

30 if (distribution_total_count == 0)
{
    if (purchase_query)
        distribution_total_count = count_set_bits(master_purchase_bitmap);
    else
        if (product_query)
            distribution_total_count = count_set_bits(master_product_bitmap);
        else
            distribution_total_count = master_count;
}

```

```

    }
    if (distribution_total_count == 0)
    {
        strcpy(msg_buffer, "\n\nNOTE:\n");
        strcat(msg_buffer, "The query you selected, could not match any records\n");
        strcat(msg_buffer, "No distribution was created. Please try another selection\n\n");
        if (DMO_CONNECTED)
            msg_buf_xfer(msg_buffer);
        else
            printf("%s", msg_buffer);

        return(SUCCESS);
    }

10  mod_buffer = file_list;
    while ((file_ptr=strcmp(mod_buffer, delimiter)) != NULL)
    {
        filename = get_string(&mod_buffer, delimiter);
        sprintf(table_name, "lookup_dir.%s.dat", filename);
        i++;
        free(filename);
        filename=NULL;
    }
15  if (file_ptr==NULL)
    {
        filename = get_string(&mod_buffer, delimiter);
        sprintf(table_name, "lookup_dir.%s.dat", filename);
        free(filename);
        filename=NULL;
    }
    i=0;

    while((tempq = next_parse_entry()) != NULL)
20  {
        if (query_tree_head == NULL)
            query_tree_head = tempq;
        else
        {
            free_parse_tree(query_tree_head);
            query_tree_head = tempq;
        }

        field = tempq->field;
25  field_number = hash(field->field_name);
        field_length = tempq->field->field_end;

        /* map section file */
        status = OpenMapFile(field->table "toolbar sec", field->field_name,
                            &chan, &retadr, field->vbn, field->number_of_blocks);
        if (is_error(status))
            error_handler(MAP_OPEN_ERR, ERROR, status, "display_sample_distribution");

30  /* set start address of data points */
        field->data->items = retadr.start;

        lookup_buffer = lookup_val_tail = NULL;

        /* search for value distribution */
        /*
        For customer distribution call the sample_distribution.
        For subsidiary distribution call the subs_distribution.
        For purchase distribution:
35  if purchase_query, then call sample_distribution.
        if not purchase_query, then call pur_pro_distribution.
        For product distribution:
        if product_query, then call sample_distribution.
        if not product_query, then call pur_pro_distribution.
        */

```



```

if (((strcmp(field->field_name, "CUS", 3)) != 0) &&
    (!purchase_query && ((strcmp(field->field_name, "PUR", 3)) == 0))) &&
    (!product_query && ((strcmp(field->field_name, "PRD", 3)) == 0))))
{
    scanf(field->field_name+3, "%d", &reference_count);

    if ((strcmp(field->field_name, "SUB", 3)) == 0)
    {
        5      load_lookup_information(&root_node, &tail_node, field->data->items, field_number, table_name,
                                     field->field_name, field->field_type, field->field_end, min_freq, max_freq,
                                     reference_count);

        distribution = show_subs_sample_distribution(field->data, &root_node,
            &tail_node, field->field_name, field_number, field->field_type,
            field->field_precision, field->field_end, master_bitmap, subsidiary_bitmap(reference_count));
    }
    else
    {
        10      if ((strcmp(field->field_name, "PUR", 3)) == 0)
                query_mode = purchase;
            else
                if ((strcmp(field->field_name, "PRD", 3)) == 0)
                    query_mode = product;

            query_type = reference_count;

        15      switch (query_type)
            {
                case avg_pur_prd:
                case total_pur_prd:
                    strcpy(msg_buffer, "\n\nNOTE:\n");
                    if (query_mode == purchase)
                        strcat(msg_buffer, "Distribution for Average or Total Purchase has not yet been implemented\n");
                    else
                        strcat(msg_buffer, "Distribution for Average or Total Product has not yet been implemented\n");
                    20      strcat(msg_buffer, "Please contact the Database Link Product Manager for release dates\n\n");
                    if (DMQ_CONNECTED)
                        msg_buf_xfer(msg_buffer);
                    else
                        printf("%s", msg_buffer);
                    return(SUCCESS);
            }

        load_lookup_information(&root_node, &tail_node, field->data->items, field_number, table_name,
            field->field_name, field->field_type, field->field_end, min_freq, max_freq,
            25      reference_count);

        distribution = show_pur_prd_sample_dist(field->data, &root_node,
            &tail_node, field->field_name, field_number, field->field_type,
            field->field_precision, field->field_end, master_bitmap, query_mode, query_type, &pur_prd_count);

        distribution_total_count = pur_prd_count;
        sprintf(temp_count_str, "%d", distribution_total_count);
        master_count_str = insert_comma(temp_count_str);

        30      if (query_mode == purchase)
            {
                sprintf(msg_buffer, "Total purchase records          %s\n", master_count_str);
                if (distribution_total_count == 0)
                {
                    35      strcat(msg_buffer, "\n\nNOTE:\n");
                            strcat(msg_buffer, "There were no purchase records found for this query.\n");
                            strcat(msg_buffer, "No distribution was created. Please try another selection.\n\n");
                            if (DMQ_CONNECTED)
                                msg_buf_xfer(msg_buffer);
                            else
                                printf("%s", msg_buffer);
                }
            }
    }
}

```

140

```

        return(SUCCESS);
    }
}
else
{
    sprintf(msg_buffer, "Total product records      : %s\n", master_count_str);
    if (distribution_total_count == 0)
    {
        5      strcat(msg_buffer, "\n\nNOTE:\n");
        strcat(msg_buffer, "There were no product records found for this query.\n");
        strcat(msg_buffer, "No distribution was created. Please try another selection.\n\n");
        if (DMO_CONNECTED)
            msg_buf_xfer(msg_buffer);
        else
            printf("%s", msg_buffer);

        10      return(SUCCESS);
    }
}

    if (DMO_CONNECTED)
        msg_buf_xfer(msg_buffer);
    else
        printf("%s\n", msg_buffer);

    15      free(master_count_str);
    master_count_str = NULL;
}
}
else
{
    if (((strcmp(field->field_name, "CUS", 3) == 0) || (strcmp(field->field_name, "SUB", 3) == 0)) &&
        (purchase_query || product_query))
    {
        20      distribution_total_count = master_count;

        sprintf(temp_count_str, "%d", master_count);
        master_count_str = insert_comma(temp_count_str);

        sprintf(msg_buffer, "Total customer records      : %s\n", master_count_str);

        if (DMO_CONNECTED)
            msg_buf_xfer(msg_buffer);
        else
            25      printf("%s\n", msg_buffer);

        free(master_count_str);
        master_count_str = NULL;
    }

    load_lookup_information(&root_node, &tail_node, field->data->items, field_number, table_name,
        field->field_name, field->field_type, field->field_end, min_freq, max_freq,
        reference_count);

    30      if ((purchase_query) && (strcmp(field->field_name, "PUR", 3) == 0))
        distribution = show_sample_distribution(field->data, &root_node,
            &tail_node, field->field_name, field_number, field->field_type,
            field->field_precision, field->field_end, master_purchase_bitmap);
    else
        if ((product_query) && (strcmp(field->field_name, "PRD", 3) == 0))
            distribution = show_sample_distribution(field->data, &root_node,
                &tail_node, field->field_name, field_number, field->field_type,
                field->field_precision, field->field_end, master_product_bitmap);
        else
            35      distribution = show_sample_distribution(field->data, &root_node,
                &tail_node, field->field_name, field_number, field->field_type,
                field->field_precision, field->field_end, master_bitmap);
    }
}

```

```

    }

    /* In case we did not specify table_names */
    if (create_table)
    {
        create_table = FALSE;
        lookup_count = 0;
    }

    /* unmap section file */
    status = UnmapCloseFile(chan, &retadr);
    if (is_error(status))
        error_handler (UNMAP_CLOSE_ERR, ERROR, status, "display_sample_distribution");

10     num_fields--;
}

/* print value distribution */
if (lookup_table_head != NULL)
{
    /* expand the field information appropriately for purchase and subsidiary */
    field_buffer = get_field_label(field->field_name field->field_label);
    /* print_sample_distribution free the various nodes as print is finished with
15     print_sample_distribution(field_buffer, field_length, lookup_table_head, num_fields, x_axis_length,
        distribution_total_count);

    free(field_buffer);
    field_buffer = NULL;
}
else
{
    strcpy(msg_buffer, "\n\nNOTE \n");
    strcat(msg_buffer, "No values found for this distribution, please try another query.\n\n");
    if (DMQ_CONNECTED)
20         msg_buf_xfer(msg_buffer);
    else
        printf("%s\n", msg_buffer);
    print_text_bufs ( );
}

/*
    Although distribution fields are build within the query_info structure, they are not queries so there is not history
    associated with that field and in addition the current_query_entry needs to be decremented to
    compensate for this
25 */
reset_query_entry(num_fields)

if (query_tree_head != NULL)
{
    free_parse_tree(query_tree_head);
    query_tree_head = NULL;
    reset_parse_tree();
}

30 /* free the lookup_buffer, this buffer gets set only for the NONE condition */
if (lookup_buffer != NULL)
{
    free_lookup_table(lookup_buffer);
    lookup_buffer = NULL;
    lookup_val_tail = NULL;
}

if (lookup_table_head != NULL)
35 {
    free_lookup_table_type(lookup_table_head);
    lookup_table_head = NULL;
    lookup_table_tail = NULL;
}

return(SUCCESS);
}

```

```

/*
  Similar to the load_lookup_data routine in the load_lookup_data, the exact value is displayed here the position is being
  displayed. This routine also builds the linked list, when a table is specified and also when no table (NONE) is specified
*/
int load_sample_count_data(root_node, root_z_node, tail_node, buffer, field_type, offset, x_axis_count, y_axis_count, z
_axis_count) struct distribution_type *root_node,
struct zposition_type *root_z_node,
struct label_type *tail_node,
5 char *buffer,
enum data_type field_type,
int offset,
int x_axis_count,
int y_axis_count,
int z_axis_count,
{
    struct lookup_table_type temp_table=NULL;
    char *temp_char;
    char *mod_buffer=NULL;
    10 char delimiter=LOOKUP_DELIMITER;
    char low_date[DATE_STRING_LENGTH], high_date[DATE_STRING_LENGTH],
    char high_char, low_char;
    int i;
    int temp_length;

    if (z_axis_count>=0)
        (z_axis_count)++;
    else
        if (y_axis_count>=0)
            (y_axis_count)++;
        15 else
            (x_axis_count)++;

    temp_table = (struct lookup_table_type *) malloc (sizeof(temp_table));
    CHECK_ALLOCATION(temp_table, "Routine load_sample_count_data, temp_table");
#ifdef DBG
    fprintf(debug_file, "%s\t%s\t%d\t%s\n", "LOAD_SAMPLE_COUNT_DATA", "TEMP_TABLE", temp_table,
    "Mallocated and initially used by lookup_
20 #endif
    temp_table->next = NULL;    table->lookup_list[offset] and then by table_tail");

    mod_buffer = buffer;
    if (strcmp(buffer, BLANK_STRING) == 0)
    {
        /*
        case where table name was not specified, continue to build the
        structure, however, don't have to build the lookup table except for OTHER
        */
        25 if (strcmp(buffer, "OTHER") == 0)
        {
            temp_table->value = malloc(FIELD_NAME_LENGTH+1);
            CHECK_ALLOCATION(temp_table->value, "Routine load_sample_count_data temp_table->value");
#ifdef DBG
            fprintf(debug_file, "%s\t%s\t%d\t%s\n", "LOAD_SAMPLE_COUNT_DATA", "TEMP_TABLE->VALUE", temp_
            table->value, "Mallocated ")
            #endif
            strcpy(temp_table->value, buffer);
            30 temp_table->low = 0;
            temp_table->high = 0;
            temp_table->pos = max_count;
            insert_sample_count_header(temp_table->value, root_node, root_z_node, tail_node, string_type,
            offset, x_axis_count, y_axis_count, z_axis_count);
        }
        else
        {
            insert_sample_count_header(BLANK_STRING, root_node, root_z_node, tail_node, string_type,
            offset, x_axis_count, y_axis_count, z_axis_count);
            35 return(1);
        }
    }
    else
    {

```

5

10

15

20

25

30

35

```
switch(field_type)
{
    case character
        temp_table->value = get_string(&mod_buffer, delimiter);
        temp_char = get_string(&mod_buffer, delimiter);
        temp_table->low = temp_char;
        free(temp_char);
        temp_char = get_string(&mod_buffer, delimiter);
        temp_table->high = temp_char;
        free(temp_char);
        temp_char = NULL;
        insert_sample_count_header(temp_table->value.root_node.root_z_node.tail_node.string_type,
                                   offset, "x_axis_count", "y_axis_count", "z_axis_count");
        break;
    case bit_type
    case small_integer
    case fixed_string
    case medium_integer
    case large_integer
    case dollars
    case floating_point
        temp_table->value = get_string(&mod_buffer, delimiter);
        temp_char = get_string(&mod_buffer, delimiter);
        temp_table->low = atoi(temp_char);
        free(temp_char);
        temp_char = get_string(&mod_buffer, delimiter);
        temp_table->high = atoi(temp_char);
        free(temp_char);
        temp_char = NULL;
        insert_sample_count_header(temp_table->value.root_node.root_z_node.tail_node.string_type,
                                   offset, "x_axis_count", "y_axis_count", "z_axis_count");
        break;
```

```

    case year_month:
    case year_month_day
        temp_table->value = get_string(&mod_buffer, delimiter);
        temp_char = get_string(&mod_buffer, delimiter);
5         temp_table->low = date_to_number(temp_char);
        free(temp_char);
        temp_char = get_string(&mod_buffer, delimiter);
        temp_table->high = date_to_number(temp_char);
        free(temp_char);
        temp_char = NULL;
        insert_sample_count_header(temp_table->value, root_node, root_z_node, tail_node, string_type,
                                   offset, *x_axis_count, *y_axis_count, *z_axis_count);

        break;
10     case string_type:
        temp_table->value = get_string(&mod_buffer, delimiter);
        temp_char = get_string(&mod_buffer, delimiter);
        temp_table->low = atoi(temp_char);
        free(temp_char);
        temp_char = get_string(&mod_buffer, delimiter);
        temp_table->high = atoi(temp_char);
        free(temp_char);
        temp_char = NULL;
        insert_sample_count_header(temp_table->value, root_node, root_z_node, tail_node, string_type,
15                                   offset, *x_axis_count, *y_axis_count, *z_axis_count);

        break;
    case bad_data_type
        return(0);
        break;
    }
}

temp_length = strlen(temp_table->value);
20 if (*z_axis_count >= 0)
{
    if (z_axis_length < temp_length)
        z_axis_length = temp_length;
}
else
    if (*y_axis_count >= 0)
    {
        if (y_axis_length < temp_length)
            y_axis_length = temp_length;
25    }
    else
        if (*x_axis_count >= 0)
        {
            if (x_axis_length < temp_length)
                x_axis_length = temp_length;
        }
    }

30 if (lookup_table->lookup_list[offset] == NULL)
{
    lookup_table->lookup_list[offset] = temp_table;
    table_tail = temp_table;
}
else
{
    table_tail->next = temp_table;
    table_tail = temp_table;
}
35 return(1);
}

```

```

/*
  This routine is called by load_sample_count_header if a table name is not specified (NONE case) It will get the values
  from the database and build the table (linked list). It will be called for both customer and subsidiary fields
*/
int get_lookup_values(value, field_name, field_type, field_precision, field_length, sub_set, temp_reference,
                    max_count, value_bitmap)
5 void
  char
enum data_type
  int
  int
  int
  unsigned int
  int
  struct bitmap
10 {
    unsigned int i, j, k;
    unsigned int jj, kk, ll, mm;
    unsigned int num_bits;
    int
    unsigned int *large_neg_integer_item;
    unsigned int *large_integer_item;
    unsigned short *medium_integer_item;
    unsigned char *small_integer_item;
15 unsigned int *bit_item;
    float *floating_item, float_num;
    double *double_item;
    char string_value[MAXCHARS];
    char *string_ptr;
    unsigned int temp_counter;
    int increment, prec_val;

    struct lookup_values *buffer, *temp, *head;
    char
20 int
    int
    int
    int
    temp_counter = value_bitmap->start;
    num_bits = value_bitmap->number_of_bits;

    kk = BITMAP_INTEGER_SIZE;
    ll = 1;
25 for (i=0; i<num_bits; i++)
    {
        if(temp_counter)
        {
            if (temp_counter & ll)
            {
                missing_value = 0;
                switch(field_type)
30 {
                    case small_integer
                        if ((*(unsigned char *) value) == MISSING_SHORT_VALUE)
                        {
                            missing_value = 1;
                        }
                        else
                        {
                            count++;
                            buffer = (struct lookup_values *) malloc(sizeof(*buffer));
                            CHECK_ALLOCATION(buffer, "Routine get_lookup_values buffer");
35 #ifdef DBG
                                fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated.");
                            #endif

```

```

        buffer->lookup_idx = (*(unsigned char *) value)
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_
        values: buffer->lookup_string");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->
        lookup_string, "Mallocated.");
        sprintf(buffer->lookup_string, "%d %d %d", buffer->lookup_idx,
        buffer->lookup_idx,
        buffer->lookup_idx);
5
        buffer->next = NULL;
    }
    break;
case character_
    if ((* (unsigned char *) value) == MISSING_SHORT_VALUE)
    {
        missing_value = 1;
    }
10
    else
    {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof("buffer"));
        CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated.");
        buffer->lookup_idx = (*(unsigned char *) value);
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");
15
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
        _string, "Mallocated.");
        buffer->lookup_idx = (*(unsigned char *) value);
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");
        sprintf(buffer->lookup_string, "%c.%c %c", buffer->lookup_idx,
        buffer->lookup_idx,
        buffer->lookup_idx);
20
        buffer->next = NULL;
    }
    break;
case dollars
    if ((* (unsigned int *) value) == MISSING_LARGE_VALUE)
    {
        missing_value = 1;
    }
25
    else
    {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof("buffer"));
        CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated.");
        buffer->lookup_idx = (*(unsigned int *) value);
        float_num = (float)buffer->lookup_idx/100.0;
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");
30
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
        _string, "Mallocated.");
        buffer->lookup_idx = (*(unsigned int *) value);
        float_num = (float)buffer->lookup_idx/100.0;
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");
        sprintf(buffer->lookup_string, "%7.2f.%d.%d", float_num,
        buffer->lookup_idx,
        buffer->lookup_idx);
35
        buffer->next = NULL;
    }
    break;
case floating_point
    if ((* (unsigned int *) value) == MISSING_LARGE_VALUE)
    {

```


147

```

        missing_value = 1;
    }
    else
    {
        prec_val = (int) pow(10, field_precision);
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Malloced ");
5 #endif

        buffer->lookup_idx = (*(unsigned int *) value);
        float_num = (float)buffer->lookup_idx/prec_val;
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
10 _string, "Malloced ");
        #endif

        sprintf(buffer->lookup_string, "%7. %d %d", field_precision float_num,
        buffer->next = NULL;
        buffer->lookup_idx
        buffer->lookup_idx);
    }
    break;
case negative_float:
    if ((*(int *) value) == MISSING_NEG_FLOAT_VALUE)
15 {
        missing_value = 1;
    }
    else
    {
        prec_val = (int) pow(10, field_precision);
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");
        buffer->lookup_idx = (*(int *) value);
        float_num = (float)buffer->lookup_idx/prec_val;
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");
20
        sprintf(buffer->lookup_string, "%7. %d %d", field_precision float_num,
        buffer->next = NULL;
        buffer->lookup_idx
        buffer->lookup_idx);
    }
    break;
case fixed_string:
25 if ((*(unsigned short *) value) == MISSING_MEDIUM_VALUE)
    {
        missing_value = 1;
    }
    else
    {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Malloced ");
30 #endif

        buffer->lookup_idx = (*(unsigned short *) value);
        field_offset = get_fixed_string_offset(field_name);
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
35 _string, "Malloced ");
        #endif

        if (fixed_field[field_offset]->fixed_string[buffer->lookup_idx] != NULL)
        {
            strcpy(buffer->lookup_string, fixed_field[field_offset]->fixed_string[buffer->lookup_idx]->strn
            sprintf(temp_buffer, "%d %d", buffer->lookup_idx, buffer->lookup_idx);

```

148

```

        strcpy(buffer->lookup_string, temp_buffer);
    }
    else
        sprintf(buffer->lookup_string, "%d.%d.%d", buffer->lookup_idx,
                                                    buffer->lookup_idx,
                                                    buffer->lookup_idx);

        buffer->next = NULL;
    }
    break;
5   case medium_integer:
        if ((*("unsigned short ") value) == MISSING_MEDIUM_VALUE)
        {
            missing_value = 1;
        }
        else
        {
            count++;
            buffer = (struct lookup_values *) malloc (sizeof("buffer"));
            10   CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");

            #ifdef DBG
                fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Malloced.");
            #endif

            buffer->lookup_idx = (*("unsigned short ") value);
            buffer->lookup_string = malloc(MAXCHARS+1);
            CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_
            values: buffer->lookup_string");
            15   #ifdef DBG
                fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->
                lookup_string, "Malloced.");
            #endif

            sprintf(buffer->lookup_string, "%d.%d.%d", buffer->lookup_idx,
                buffer->lookup_idx,
                buffer->lookup_idx);
            buffer->next = NULL;
        }
        break;
    case bit_type:
    case large_integer:
        if ((*("unsigned int ") value) == MISSING_LARGE_VALUE)
        {
            missing_value = 1;
        }
        else
        {
            count++;
            buffer = (struct lookup_values *) malloc (sizeof("buffer"));
            25   CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");

            #ifdef DBG
                fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Malloced.");
            #endif

            buffer->lookup_idx = (*("unsigned int ") value);
            buffer->lookup_string = malloc(MAXCHARS+1);
            CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values
            buffer->lookup_string");
            30   #ifdef DBG
                fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup_
                string, "Malloced.")
            #endif

            sprintf(buffer->lookup_string, "%d.%d.%d", buffer->lookup_idx,
                buffer->lookup_idx,
                buffer->lookup_idx);
            buffer->next = NULL;
        }
        break;
    case year_month:
    case year_month_day:
        if ((*("unsigned short ") value) == MISSING_MEDIUM_VALUE)
        {
            missing_value = 1;
        }

```

```

    }
    else
    {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");

        #ifdef DBG
        fprintf(debug_file, "%s\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated");
        #endif

        buffer->lookup_idx = ((unsigned short *) value);
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");

        #ifdef DBG
        fprintf(debug_file, "%s\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup_string, "Mallocated");
        #endif

        sprintf(buffer->lookup_string, "%s.%s.%s", number_to_date(buffer->lookup_idx),
                                                    number_to_date(buffer->lookup_idx),
                                                    number_to_date(buffer->lookup_idx));

        buffer->next = NULL;
    }
    break;
case string_type:
    strcpy(string_value, (unsigned char *) value, field_length);
    string_value[field_length] = '\0';
    if (strcmp(string_value, MISSING_STRING_VALUE, field_length) == 0)
    {
        missing_value = 1;
    }
    else
    {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_lookup_values: buffer");

        #ifdef DBG
        fprintf(debug_file, "%s\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated");
        #endif

        buffer->lookup_string = malloc(MAXCHARS+1);

        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_lookup_values: buffer->lookup_string");

        #ifdef DBG
        fprintf(debug_file, "%s\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup_string, "Mallocated");
        #endif

        strcpy(buffer->lookup_string, string_value, field_length);
        buffer->lookup_string[field_length] = '\0';
        buffer->lookup_idx = hash_xchar(buffer->lookup_string, field_length);
        sprintf(temp_buffer, "%c.%d", buffer->lookup_idx, buffer->lookup_idx);
        strcat(buffer->lookup_string, temp_buffer);
        buffer->next = NULL;
    }
    break;
case bad_data_type:
    return(0);
    break;
}

if (!missing_value)
{
    /* include this with the structure */
    if (lookup_buffer == NULL)
    {
        lookup_buffer = buffer;
        lookup_val_tail = buffer;
    }
    else
    {

```

```

/* check where to insert this */
temp = lookup_buffer;
head = lookup_buffer;
while (temp != NULL)
{
    if (count >= max_count)
        return(1);

    if (field_type == string_type)
    {
        if ((status = strcmp(buffer->lookup_string, temp->lookup_string)) < 0)
        {
            if (count < max_count)
            {
                if (temp == head)
                {
                    buffer->next = lookup_buffer;
                    lookup_buffer = buffer;
                }
                else
                {
                    buffer->next = temp;
                    head->next = buffer;
                }

                if ((count+1) == max_count)
                    return(1);
                break;
            }
            else
                return(1);
        }
        else
        {
            if (status == 0)
            {
                -count;
            }
        }
    }

    #ifdef DBG
        fprintf(debug_file, "%s/%s/%d/%s\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING",
            buffer->lookup_string, "Freed");
    #endif
        free(buffer->lookup_string);
        buffer->lookup_string = NULL;

    #ifdef DBG
        fprintf(debug_file, "%s/%s/%d/%s\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Freed");
    #endif

        free(buffer);
        buffer = NULL;
        break;
    }
}

}
else
{
    if (buffer->lookup_idx < temp->lookup_idx)
    {
        if (count < max_count)
        {
            if (temp == head)
            {
                buffer->next = lookup_buffer;
                lookup_buffer = buffer;
            }
            else
            {

```

```

        buffer->next = temp;
        head->next = buffer;
    }

    if ((count+1) == max_count)
        return(1);

5      break;
    }
    else
        return(1);
    }
    else
        if (buffer->lookup_idx == temp->lookup_idx)
        {
            -count;
10
        }

#ifdef DBG
    fprintf(debug_file, "%s\n%s\n%d\n%s\n", "GET_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING",
        buffer->lookup_string, "Freed ");
#endif

    free(buffer->lookup_string);
    buffer->lookup_string = NULL;
15
#ifdef DBG
    fprintf(debug_file, "%s\n%s\n%d\n%s\n", "GET_LOOKUP_VALUES", "BUFFER", buffer, "Freed ");
#endif

    free(buffer);
    buffer = NULL;
    break;
}

head = temp;
temp = temp->next;
}

if (temp == NULL)
{
    if (count < max_count)
    {
        lookup_val_tail->next = buffer;
        lookup_val_tail = buffer;

        if ((count+1) == max_count)
            return(1);
    }
    else
        return(1);
}
}

}

30      increment = 1;
      if (sub_set)
          if (temp_reference & ij)
              temp_reference++;
          else
              increment = 0;

      if (increment)
      {
35          switch(field_type)
          {
              case dollars:
              case floating_point:
              case large_integer:
              case bn_type:

```

```

        large_integer_item = (unsigned int *) value;
        large_integer_item++;
        value = (void *) large_integer_item;
        break;
5      case negative_float:
        large_neg_integer_item = (int *) value;
        large_neg_integer_item++;
        value = (void *) large_neg_integer_item;
        break;
      case fixed_string:
      case year_month:
      case year_month_day:
      case medium_integer:
10        medium_integer_item = (unsigned short *) value;
        medium_integer_item++;
        value = (void *) medium_integer_item;
        break;
      case character:
      case small_integer:
        small_integer_item = (unsigned char *) value;
        small_integer_item++;
        value = (void *) small_integer_item;
        break;
15      case string_type:
        string_ptr = (unsigned char *) value;
        string_ptr += field_length;
        value = (void *) string_ptr;
        break;
    }
  }

  if ( !kk )
20    jj <= 1
  else
  {
    temp_counter++
    jj = 1
    kk = BITMAP_INTEGER_SIZE;
  }
}
/* temp_counter == 0 bump past whole word */
else
25 {
  if (sub_set)
  {
    if (temp_reference)
    {
      for (ll=0,mm=1, ll<BITMAP_INTEGER_SIZE, ll++,mm<=1)
      {
        if (temp_reference & mm)
30 {
          switch(field_type)
          {
            case dollars:
            case floating_point:
            case large_integer:
            case bit_type:
              large_integer_item = (unsigned int *) value;
              large_integer_item++;
              value = (void *) large_integer_item;
              break;
            case negative_float:
            case large_neg_integer_item = (int *) value;
            case large_neg_integer_item++;
            case value = (void *) large_neg_integer_item;
            case break;
35

```

```

5
    case fixed_string
    case year_month:
    case year_month_day:
    case medium_integer:
        medium_integer_item = (unsigned short *) value;
        medium_integer_item += BITMAP_INTEGER_SIZE;
        value = (void *) medium_integer_item;
        break;
    case character
    case small_integer
        small_integer_item = (unsigned char *) value;
        small_integer_item += BITMAP_INTEGER_SIZE;
        value = (void *) small_integer_item;
        break;
    case string_type:
        string_ptr = (unsigned char *) value;
        string_ptr += field_length;
        value = (void *) string_ptr;
        break;
    }
    temp_reference++;
15
    }
    else
    {
        switch(field_type)
        {
            case dollars
            case floating_point
            case large_integer
            case bit_type
                large_integer_item = (unsigned int *) value;
                large_integer_item += BITMAP_INTEGER_SIZE;
                value = (void *) large_integer_item;
                break;
            case negative_float
                large_neg_integer_item = (int *) value;
                large_neg_integer_item += BITMAP_INTEGER_SIZE;
                value = (void *) large_neg_integer_item;
                break;
            case fixed_string
            case year_month
            case year_month_day
            case medium_integer
                medium_integer_item = (unsigned short *) value;
                medium_integer_item += BITMAP_INTEGER_SIZE;
                value = (void *) medium_integer_item;
                break;
            case character
            case small_integer
                small_integer_item = (unsigned char *) value;
                small_integer_item += BITMAP_INTEGER_SIZE;
                value = (void *) small_integer_item;
                break;
            case string_type
                string_ptr = (unsigned char *) value;
                string_ptr += (field_length * BITMAP_INTEGER_SIZE);
                value = (void *) string_ptr;
                break;
        }
    }
    i += BITMAP_INTEGER_SIZE - 1;
    temp_counter++;
35
    }
    }

```

```

/*
  This routine is called by load_sample_count_header if a table name is not specified (NONE case).
  It will get the values from the database and build the table (linked list). It will be called for both purchase
  and product fields.
*/
5 int get_mf_lookup_values(value, field_name, field_type, field_precision, field_length, reference_map,
                           max_count, value_bitmap)
    void
    char
    enum date_type
    int
    int
    unsigned short
    int
    struct bitmap
10 {
    unsigned int i, j, k, m;
    unsigned int j, kk, ll, mm;
    unsigned int num_bits;
    int
    unsigned int *large_neg_integer_rem;
    unsigned int *large_integer_rem;
    unsigned short *medium_integer_rem;
    unsigned char *small_integer_rem;
15 unsigned int *bit_rem;
    float *floating_rem, float_num;
    double *double_rem;
    char string_value[MAXCHARS];
    char *string_ptr;
    unsigned int temp_counter;
    int
    prec_val;

    struct lookup_values *buffer, *temp, *head;
20 char
    int
    int
    int
    int
    temp_buffer[MAXCHARS];
    count = -1;
    field_offset;
    missing_value = 0;
    status = 0;

    temp_counter = value_bitmap->start;
    num_bits = value_bitmap->number_of_bits;

    kk = BITMAP_INTEGER_SIZE;
25 j = 1;

    for (i=0; i<num_bits; i++)
    {
        if (*temp_counter)
        {
            if (*reference_map)
            {
                if (*temp_counter & j)
                {
30 for (m = 0; m < *reference_map; m++)
                {
                    missing_value = 0;
                    switch(field_type)
                    {
                        case small_integer:
                            if ((*((unsigned char *) value)) == MISSING_SHORT_VALUE)
                            {
                                missing_value = 1;
35 }
                            else
                            {
                                count++;
                                buffer = (struct lookup_values *) malloc(sizeof(*buffer));
                                CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values: buffer");

```



```

#ifdef DBG
    fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated ");
#endif

    buffer->lookup_idx = ((unsigned char *) value+m);
    buffer->lookup_string = malloc(MAXCHARS+1);
    CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values: buffer->lookup_string");

#ifdef DBG
    fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
5      _string, "Mallocated ");
#endif

    sprintf(buffer->lookup_string, "%d.%d.%d", buffer->lookup_idx,
                                buffer->lookup_idx,
                                buffer->lookup_idx);
    buffer->next = NULL;
}
break;
10 case character
    if (((unsigned char *) value) == MISSING_SHORT_VALUE)
    {
        missing_value = 1;
    }
    else
    {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof("buffer"));
        CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values: buffer");
15
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated ");
#endif

        buffer->lookup_idx = ((unsigned char *) value+m);
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values: buffer->lookup_string");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
20      _string, "Mallocated ");
#endif

        sprintf(buffer->lookup_string, "%c.%c.%c", buffer->lookup_idx,
                                buffer->lookup_idx,
                                buffer->lookup_idx);
        buffer->next = NULL;
    }
    break;
case dollars
25
    if (((unsigned int *) value) == MISSING_LARGE_VALUE)
    {
        missing_value = 1;
    }
    else
    {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof("buffer"));
        CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values: buffer");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated ");
#endif
30
        buffer->lookup_idx = ((unsigned int *) value+m);
        float_num = (float)buffer->lookup_idx/100.0;
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values: buffer->lookup_string");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
35      _string, "Mallocated ");
#endif

        sprintf(buffer->lookup_string, "%7.2f.%d.%d", float_num,
                                buffer->lookup_idx,
                                buffer->lookup_idx);
        buffer->next = NULL;
    }

```

```

    }
    break;
case floating_point:
    if (((unsigned int *) value) == MISSING_LARGE_VALUE)
    {
        missing_value = 1;
    }
    else
    {
5       prec_val = (int) pow(10, field_precision);
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values buffer");
        #ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Malloced");
        #endif

        buffer->lookup_idx = (((unsigned int *) value)+m);
        float_num = (float)buffer->lookup_idx/prec_val;
        buffer->lookup_string = malloc(MAXCHARS+1);
10      CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values buffer->lookup_string");
        #ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
            _string, "Malloced");
        #endif

        sprintf(buffer->lookup_string, "%7.1f,%d,%d", field_precision float_num,
                                                    buffer->lookup_idx,
                                                    buffer->lookup_idx);

        buffer->next = NULL;
15      }
        break;
case negative_float:
    if (((int *) value) == MISSING_NEG_FLOAT_VALUE)
    {
        missing_value = 1;
    }
    else
    {
20      prec_val = (int) pow(10, field_precision);
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values buffer");
        buffer->lookup_idx = (((int *) value)+m);
        float_num = (float)buffer->lookup_idx/prec_val;
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values buffer->lookup_string");

        sprintf(buffer->lookup_string, "%7.1f,%d,%d", field_precision float_num,
                                                    buffer->lookup_idx,
25      buffer->lookup_idx);

        buffer->next = NULL;
    }
    break;
case fixed_string:
    if (((unsigned short *) value) == MISSING_MEDIUM_VALUE)
    {
        missing_value = 1;
    }
    else
30      {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values: buffer");
        #ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Malloced");
        #endif

        buffer->lookup_idx = (((unsigned short *) value)+m);
        field_offset = get_fixed_string_offset(field_name);
35      buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values: buffer->lookup_string");
        #ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup
            _string, "Malloced");
        #endif

        if (fixed_field[field_offset] fixed_string[buffer->lookup_idx] != NULL)

```

157

```

    {
        strcpy(buffer->lookup_string, fixed_field(field_offset)->fixed_string(buffer->lookup_idx)->string);
        sprintf(temp_buffer, "%d,%d", buffer->lookup_idx, buffer->lookup_idx);
        strcat(buffer->lookup_string, temp_buffer);
    }
    else
        sprintf(buffer->lookup_string, "%d,%d,%d", buffer->lookup_idx,
                                                    buffer->lookup_idx,
                                                    buffer->lookup_idx);

5      buffer->next = NULL;
    }
    break;
case medium_integer
    if (((unsigned short *) value) == MISSING_MEDIUM_VALUE)
    {
        missing_value = 1;
    }
    else
10     {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values: buffer");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Malloced ");
        #endif

        buffer->lookup_idx = (((unsigned short *) value)+m);
        buffer->lookup_string = malloc(MAXCHARS+1);
15     CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values: buffer->lookup_string");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING",
            buffer->lookup_string, "Malloced.");
        #endif

        sprintf(buffer->lookup_string, "%d,%d,%d", buffer->lookup_idx,
            buffer->lookup_idx,
            buffer->lookup_idx);
            buffer->next = NULL;
20     }
    break;
case bit_type
case large_integer
    if (((unsigned int *) value) == MISSING_LARGE_VALUE)
    {
        missing_value = 1;
    }
    else
25     {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof(*buffer));
        CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values: buffer");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Malloced.");
        #endif

        buffer->lookup_idx = (((unsigned int *) value)+m);
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values:
30     buffer->lookup_string");
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->
            lookup_string, "Malloced.");
        #endif

        sprintf(buffer->lookup_string, "%d,%d,%d", buffer->lookup_idx,
                                                    buffer->lookup_idx,
                                                    buffer->lookup_idx);
35     buffer->next = NULL;
    }
    break;
case year_month:
case year_month_day:
    if (((unsigned short *) value) == MISSING_MEDIUM_VALUE)
    {
        missing_value = 1;
    }

```

158

```

    }
    else
    {
        count++;
        buffer = (struct lookup_values *) malloc (sizeof("buffer"));
        CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values: buffer");

#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated ");
#endif

        buffer->lookup_idx = ((unsigned short *) value+m);
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values: buffer->lookup_

5
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING",
        buffer->lookup_string, "Mallocated ");
#endif

        sprintf(buffer->lookup_string, "%s, %s, %s", number_to_date(buffer->lookup_idx),
        number_to_date(buffer->lookup_idx),
        number_to_date(buffer->lookup_idx);
        buffer->next = NULL;

10
    }
    break;
    case string_type
        buffer->lookup_string = malloc(MAXCHARS+1);
        CHECK_ALLOCATION(buffer->lookup_string, "Routine get_mf_lookup_values: buffer->lookup_string

#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING", buffer->lookup_string,
        "Mallocated ");
15
#endif

        strcpy(string_value, ((unsigned char *) value+m), field_length);
        string_value[field_length] = '\0';
        if (strcmp(string_value, MISSING_STRING_VALUE, field_length) == 0)
        {
            missing_value = 1;
        }
        else

20
        {
            count++;
            buffer = (struct lookup_values *) malloc (sizeof("buffer"));
            CHECK_ALLOCATION(buffer, "Routine get_mf_lookup_values: buffer");

#ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Mallocated ");
            sendf

25
            strcpy(buffer->lookup_string, string_value, field_length);
            buffer->lookup_string[field_length] = '\0';
            buffer->lookup_idx = hash_xchar(buffer->lookup_string, field_length);
            sprintf(temp_buffer, "%d %d", buffer->lookup_idx, buffer->lookup_idx);
            strcat(buffer->lookup_string, temp_buffer);
            buffer->next = NULL;

        }
        break;
    case bad_data_type
        return(0);
        break;
    }

30
    if (!missing_value)
    {
        /* include this with the structure */
        if (lookup_buffer == NULL)
        {
            lookup_buffer = buffer;
            lookup_val_tad = buffer;
        }
        else

35
        {
            /* check where to insert this */
            temp = lookup_buffer;
            head = lookup_buffer;
            while (temp != NULL)
            {
                if (count >= max_count)
                    return(1);
            }
        }
    }

```

```

if (field_type == string_type)
{
    if ((status = strcmp(buffer->lookup_string, temp->lookup_string)) < 0)
    {
        if (count < max_count)
        {
            if (temp == head)
            {
                buffer->next = lookup_buffer;
                lookup_buffer = buffer;
            }
            else
            {
                buffer->next = temp;
                head->next = buffer;
            }

            if ((count+1) == max_count)
                return(1);

            break;
        }
        else
            return(1);
    }
    else
    {
        if (status == 0)
        {
            -count;

            #ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING",
            buffer->lookup_string, "Freed") #endif

            free(buffer->lookup_string);
            buffer->lookup_string = NULL;

            #ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Freed")
            #endif

            free(buffer);
            buffer = NULL;
            break;
        }
    }
}
else
{
    if (buffer->lookup_idx < temp->lookup_idx)
    {
        if (count < max_count)
        {
            if (temp == head)
            {
                buffer->next = lookup_buffer;
                lookup_buffer = buffer;
            }
            else
            {
                buffer->next = temp;
                head->next = buffer;
            }

            if ((count+1) == max_count)

```

```

        return(1);
        break;
    }
    else
        return(1);
5   }
    else
        if (buffer->lookup_idx == temp->lookup_idx)
        {
            --count;

#ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER->LOOKUP_STRING",
10             buffer->lookup_string, "Free");
#endif

            free(buffer->lookup_string);
            buffer->lookup_string = NULL;

#ifdef DBG
            fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "GET_MF_LOOKUP_VALUES", "BUFFER", buffer, "Free");
            #endif

            free(buffer);
            buffer = NULL;
            break;
        }
15     head = temp;
        temp = temp->next;
    }

    if (temp == NULL)
    {
        if (count < max_count)
        {
20             lookup_val_tail->next = buffer;
            lookup_val_tail = buffer;

            if ((count+1) == max_count)
                return(1);
        }
        else
            return(1);
    }
}

25 }

switch(field_type)
{
    case dollars:
    case floating_point:
    case large_integer:
    case bit_type:
30     large_integer_item = (unsigned int *) value;
        large_integer_item += *reference_map;
        value = (void *) large_integer_item;
        break;

    case negative_float:
        large_neg_integer_item = (int *) value;
        large_neg_integer_item += *reference_map;
        value = (void *) large_neg_integer_item;
        break;

    case fixed_string:
35     case year_month:
    case year_month_day:
    case medium_integer:
        medium_integer_item = (unsigned short *) value;
        medium_integer_item += *reference_map;
        value = (void *) medium_integer_item;
        break;
}

```

```

    case character:
    case small_integer
        small_integer_item = (unsigned char *) value;
        small_integer_item += *reference_map;
        value = (void *) small_integer_item;
        break;
    case string_type:
        string_ptr = (unsigned char *) value;
        string_ptr += (*reference_map * field_length);
        value = (void *) string_ptr;
        break;
    }
    reference_map++;
}
else
    reference_map++;

if ( -kk )
    ll <= 1;
else
{
    temp_counter++;
    ll = 1;
    kk = BITMAP_INTEGER_SIZE;
}
}
/* temp_counter == 0, bump past whole word */
else
{
    for (ll=0; ll<BITMAP_INTEGER_SIZE; ll++)
    {
        switch(field_type)
        {
            case dollars:
            case floating_point:
            case large_integer
            case bit_type
                large_integer_item = (unsigned int *) value;
                large_integer_item += *reference_map;
                value = (void *) large_integer_item;
                break;
            case negative_float:
                large_neg_integer_item = (int *) value;
                large_neg_integer_item += *reference_map;
                value = (void *) large_neg_integer_item;
                break;
            case fixed_string:
            case year_month:
            case year_month_day:
            case medium_integer
                medium_integer_item = (unsigned short *) value;
                medium_integer_item += *reference_map;
                value = (void *) medium_integer_item;
                break;
            case character:
            case small_integer
                small_integer_item = (unsigned char *) value;
                small_integer_item += *reference_map;
                value = (void *) small_integer_item;
                break;
            case string_type:
                string_ptr = (unsigned char *) value;
                string_ptr += (*reference_map * field_length);
                value = (void *) string_ptr;
                break;
        }
    }
}

```

```

        )
        reference_map++;
    }
    i += BITMAP_INTEGER_SIZE - 1;
    temp_counter++;
}

/*
5 This routine similar to the load_lookup_information. The difference being load_lookup_information
handles a single field distribution, this handles multiple field distribution. It builds the lookup table
(linked list). For a distribution, a table name (filename) or NONE can be specified. If a table name is
specified the entries of that file are build as linked list. However, for the NONE case, get_lookup_
values or get_mi_lookup values is called to get the values and the lookup_table is then built.
*/
load_sample_count_header(root_node, root_z_node, tail_node, table_name, value, field_name, field_type, field
precision, field_length, sub_set, temp_reference, mi_case_flag, offset, x_axis_count, y_axis_count, z_axis
count, reference_bitmap)
10 struct distribution_type *root_node;
struct zposition_type *root_z_node;
struct label_type *tail_node;
char *table_name;
void *value;
char *field_name;
enum data_type field_type;
int field_precision;
int field_length;
int sub_set;
15 unsigned int temp_reference;
int mi_case_flag;
int offset;
int x_axis_count;
int y_axis_count;
int z_axis_count;
struct bitmap reference_bitmap;
{
    FILE *f;
    char *mod_buffer, *temp_string, *file_ptr;
    char buffer[MAXCHARS];
    struct lookup_values *temp_lookup_buffer;
    struct lookup_table_list_type *lookup_sample_count_table;
    struct lookup_table_type *temp_table;
    int i, low, high, projection, difference = 10;
    char *temp_table_name;
    unsigned short *reference_map;

    if (lookup_table == NULL)
25 {
        lookup_table = (struct lookup_table_list_type *) malloc (sizeof(lookup_table));
        CHECK_ALLOCATION(lookup_table, "Routine load_sample_count_header lookup_table");
#ifdef DBG
        fprintf(debug_file, "%s\n", "LOAD_SAMPLE_COUNT_HEADER", "LOOKUP_TABLE", lookup_table
            "Malloced");
#endif
        for (i = 0; i < MAXFIELDS; i++)
            lookup_table->lookup_list[i] = NULL;
    }
    /* check if table name exists */
    if (table_name == NULL)
        return(0);
    else
    {
        temp_table_name = strtoupper(table_name);
        if (strcmp(temp_table_name, "LOOKUP_DIR:NONE.DAT", 20) == 0)
        {
            /* for the OTHER case */
            if (z_axis_length < 5)
            35 {
                z_axis_length = 5;
            }
            if (y_axis_length < 5)
            {
                y_axis_length = 5;
            }
            if (x_axis_length < 5)
            {
                x_axis_length = 5;
            }
            /* handle for the OTHER Case */
            if (lookup_table != NULL)
            {

```



```

    if (~z_axis_count >= 0)
        create_z_table = TRUE.
    else
        if (~y_axis_count >= 0)
            create_y_table = TRUE.
        else
            if (~x_axis_count >= 0)
                create_x_table = TRUE.
    max_count = 0.

    /* svp - assuming that we dont want to print larger than 10 values in the x-axis */
    if (create_z_table)
        max_count = MAX_Z_AXIS_COUNT.
    else
        if (create_y_table)
            max_count = MAX_Y_AXIS_COUNT.
        else
            if (create_x_table)
                max_count = MAX_X_AXIS_COUNT.

    /*
    DONT NEED THIS
    if (max_count == 0)
    {
        if (master_count > MAX_LOOKUP_COUNT_VALUE)
            max_count = MAX_LOOKUP_COUNT_VALUE.
        else
            max_count = master_count.
    }
    */

    /* get lookup_values from the database */
    /* mf_case_flag is set. if we are doing the purchase distribution on a non-purchase bitmap */
    /* or a product distribution on a non-product bitmap */
    if (mf_case_flag)
    {
        if (strcmp(field_name, "PUR") == 0)
            reference_map = pur01_map;
        else
            reference_map = prd01_map;

        get_mf_lookup_values(value, field_name, field_type, field_precision, field_length,
            reference_map, max_count, reference_bitmap).
    }
    else
        get_lookup_values(value, field_name, field_type, field_precision, field_length, sub_sel,
            temp_reference, max_count, reference_bitmap);

    /* include OTHER as the last selection */
    temp_lookup_buffer = (struct lookup_values *) malloc (sizeof(temp_lookup_buffer));
    CHECK_ALLOCATION(temp_lookup_buffer, "Routine load_sample_count_header: temp_lookup_buffer");

#ifdef DBG
30 fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "LOAD_SAMPLE_COUNT_HEADER", "TEMP_LOOKUP_BUFFER", temp_
    lookup_buffer, "Malloced");
#endif
    temp_lookup_buffer->lookup_idx = MISSING_LARGE_VALUE;
    temp_lookup_buffer->lookup_string = malloc(MAXCHARS+1);
    CHECK_ALLOCATION(temp_lookup_buffer->lookup_string, "Routine load_sample_count_header: temp_lookup_
    buffer->lookup_string");
#ifdef DBG
    fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "LOAD_SAMPLE_COUNT_HEADER", "TEMP_LOOKUP_BUFFER
35 ->LOOKUP_STRING",
    temp_lookup_buffer->lookup_string);
#endif
    sprintf(temp_lookup_buffer->lookup_string, "%s,%d,%d", "OTHER",
    0,
    temp_lookup_buffer->lookup_idx);
    temp_lookup_buffer->next = NULL;

    /* join with existing structure */
    if (lookup_buffer == NULL)
    {
        lookup_buffer = temp_lookup_buffer;
        lookup_val_tail = temp_lookup_buffer;
    }

```

```

else
{
    lookup_val_tail->next = temp_lookup_buffer;
    lookup_val_tail = temp_lookup_buffer;
}

temp_lookup_buffer = lookup_buffer;
while (temp_lookup_buffer != NULL)
5 {
    load_sample_count_data(root_node, root_z_node, tail_node, temp_lookup_buffer->lookup_string, field_type,
                           offset, x_axis_count, y_axis_count, z_axis_count);
    temp_lookup_buffer = temp_lookup_buffer->next;
}

/* free structures */
free(temp_table_name);
temp_table_name = NULL;
10 return(1);
}

/* free structures, if it is not a NONE condition */
free(temp_table_name);
temp_table_name = NULL;
}

15 mod_buffer = table_name;
if ((file_ptr = strchr(table_name, '.')) != NULL)
{
    temp_string = get_string(&mod_buffer, '.');
    sscanf(temp_string, "%d", &low);
    free(temp_string);
    temp_string = NULL;
    sscanf(mod_buffer, "%d", &high);
    if ((int)(high/difference) <= 1)
20 difference = 1;
    if ((projection = (int)(high/difference)) <= 0)
        projection++;
    for (i = low; i < projection; i++)
    {
        sprintf(buffer, "%d+ %d %d", low, low, (low+difference));
        load_sample_count_data(root_node, root_z_node, tail_node, buffer, field_type,
                               offset, x_axis_count, y_axis_count, z_axis_count);
        low += (difference+1);
25 }
    }
else
{
    if ((ff = fopen(table_name, "r")) != NULL)
    {
        while (fgets(buffer, MAXCHARS, ff) != NULL)
        {
30 load_sample_count_data(root_node, root_z_node, tail_node, buffer, field_type,
                           offset, x_axis_count, y_axis_count, z_axis_count);
        }
    }
    else
    {
        error_handler(FILE_NOT_OPEN, ERROR, NO_STATUS, table_name);
    }

35 if (fclose(ff) == EOF)
    error_handler(FILE_NOT_CLOSE, ERROR, NO_STATUS, table_name);
}

```

```

        if ("z_axis_count">=0)
            ("z_axis_count")++;
        else
            if ("y_axis_count">=0)
                ("y_axis_count")++;
            else
                ("x_axis_count")++;

        temp_table = (struct lookup_table_type *) malloc (sizeof(*temp_table));
        CHECK_ALLOCATION(temp_table, "Routine load_sample_count_header: temp_table");
#ifdef DBG
        fprintf(debug_file, "%s\n", "LOAD_SAMPLE_COUNT_HEADER", "TEMP_TABLE", temp_table, "Mallocated");
#endif
        temp_table->next = NULL;

10     temp_table->value = malloc(FIELD_NAME_LENGTH+1);
        CHECK_ALLOCATION(temp_table->value, "temp_table->value:load_sample_count_header()");
#ifdef DBG
        fprintf(debug_file, "%s\n", "LOAD_SAMPLE_COUNT_HEADER", "TEMP_TABLE->VALUE", temp_table->value, "Mallocated");
#endif
        strcpy(temp_table->value, "OTHER", 5);
        temp_table->value[5] = '\0';
        temp_table->low = LOW_VALUE;
        temp_table->high = HIGH_VALUE;
15     insert_sample_count_header(temp_table->value.root_node.root_z_node.tail_node.string_type,
                                offset, "x_axis_count", "y_axis_count", "z_axis_count");

        table_tail->next = temp_table;
        table_tail = temp_table;
    }
    else
    {
20         printf("Error occurred while creating the lookup table.\n");
        return(0);
    }
}

/*
Not using this routine, it can be deleted
If a table name is not specified (NONE case) for a distribution field the appropriate axis variable (create_x_table, . . )
is set. This routine checks to see if that variable is set based on the field offset, and returns a TRUE or FALSE
*/
int get_table_creation_status (offset)
int offset;
{
    int table_status = FALSE;

    switch(offset)
    {
30         case 0 : if (create_x_table)
                    table_status = TRUE;
                    break;
        case 1 : if (create_y_table)
                    table_status = TRUE;
                    break;
        case 2 : if (create_z_table)
                    table_status = TRUE;
                    break;
    }
35     return(table_status);
}

```

```

    /*
    This routine returns the position of the OTHER bucket. This is important if my list limit is exhausted and I want to
    put the new value in the OTHER bucket.
    */
    int get_other_count(temp_table)
    struct lookup_table_type *temp_table;
    {
        int status = 0;

        while (temp_table != NULL)
        {
            if ((strcmp(temp_table->value, "OTHER", 5)) == 0)
                return(status);
            status++;
            temp_table = temp_table->next;
        }
        return(status);
    }

    /*
    Every value read from the database is compared to the table of values (linked list). If a value is found the counter value
    is passed back, signifying the position of the value. If the value is not found but less than the first table entry, the
    position of the OTHER bucket is passed back.

    Special_Case: For purchase or product distribution (X2 or X3) on a customer query, for every bit set on the customer
    bitmap all purchases for that customer is being captured. The "incr" value goes from 0 to
    the map_count value for that customer to capture the scenario of all purchases.
    */
    int get_bucket_number(search_value, lookup_bucket_table, offset, field_name, field_type, field_length, incr,
        root_node)
    void *search_value;
    struct lookup_table_type *lookup_bucket_table;
    int offset;
    char *field_name;
    enum data_type field_type;
    int field_length;
    int incr;
    struct distribution_type *root_node;
    {
        struct lookup_table_type *temp_table;
        int pos = 0;
        int status;
        int max_value_count;
        int count = 0;
        int table_status = FALSE;

        temp_table = *lookup_bucket_table;

        if ((offset == 0) && (create_x_table))
        {
            table_status = TRUE;
            max_value_count = MAX_X_AXIS_COUNT;
        }
        else
        {
            if ((offset == 1) && (create_y_table))
            {
                table_status = TRUE;
                max_value_count = MAX_Y_AXIS_COUNT;
            }
            else
            {
                if ((offset == 2) && (create_z_table))
                {

```

```

        table_status = TRUE;
        max_value_count = MAX_Z_AXIS_COUNT;
    }
    else
    {
        if (offset == 0)
        {
            if (max_col_value_count == 0)
                max_value_count =
                max_col_value_count = get_other_count(temp_table);
            else
                max_value_count = max_col_value_count;
        }
        else
        {
            if (offset == 1)
            {
                if (max_row_value_count == 0)
                    max_value_count =
                    max_row_value_count = get_other_count(temp_table);
                else
                    max_value_count = max_row_value_count;
            }
            else
            {
                if (offset == 2)
                {
                    if (max_page_value_count == 0)
                        max_value_count =
                        max_page_value_count = get_other_count(temp_table);
                    else
                        max_value_count = max_page_value_count;
                }
            }
        }
    }
}

/* check to find which of the tables were dynamically created (table name of NONE) */
/* table_status = get_table_creation_status(offset); */

if (*lookup_bucket_table == NULL)
    return(-1);

while (temp_table != NULL)
{
    /* check if value exists */
    switch(field_type)
    {
        case dollars
        case floating_point
        case large_integer
        case bit_type
    }
    if (((unsigned int *) search_value + incr) temp_table->low)

```

```

        {
            if ( table_status )
                return(max_value_count);
            else
                break;
        }
5      if ( *((unsigned int *) search_value + incr) <= temp_table->high )
        return(pos);
        break;
    case negative_float:
        if ( *((int *) search_value + incr) < temp_table->low )
        {
            if ( table_status )
10             return(max_value_count);
            else
                break;
        }
        if ( *((int *) search_value + incr) <= temp_table->high )
            return(pos);
        break;
    case small_integer:
    case character:
15      if ( *((unsigned char *) search_value + incr) < temp_table->low )
        {
            if ( table_status )
                return(max_value_count);
            else
                break;
        }
        if ( *((unsigned char *) search_value + incr) <= temp_table->high )
            return(pos);
        break;
20    case fixed_string
    case medium_integer
    case year_month
    case year_month_day
        if ( *((unsigned short *) search_value + incr) < temp_table->low )
        {
            if ( table_status )
                return(max_value_count);
            else
                break;
        }
25      if ( *((unsigned short *) search_value + incr) <= temp_table->high )
        return(pos);
        break;
    case string_type
        status = strcmp( *((unsigned char *) search_value + incr), temp_table->value field_length);
        if (status == 0)
            return(pos);
        else
            if ((status < 0) && table_status)
30             return(max_value_count);
        break;
    case bad_data_type
        break;
    }

    pos++;

    temp_table = temp_table->next;
35  return(pos-1);
}

```

Actually builds the distribution (X2 or X3). This routine will be called either for a customer distribution, or for a subsidiary, or for a purchase/product distribution ONLY IF a purchase/product query was found. The routine captures the row, column, and page position and updates the total for that array value.

```

5 struct distribution_type *show_sample_count_distribution(value, root_node, tail_node, field_name, field_type,
   field_precision, field_length, sub_set, temp_reference,
   num_fields, lookup_table, value_bitmap)

void
   struct distribution_type *root_node;
   struct value_node *tail_node;
   char *field_name[];
10   enum data_type field_type[];
   int field_precision[];
   unsigned int field_length[];
   unsigned int sub_set[];
   unsigned int temp_reference[];
   int num_fields;
   struct lookup_table_list_type *lookup_table;
   struct bitmap *value_bitmap;
   {
15       int row, column, position;
       unsigned int i, j, k;
       unsigned int mcr=0;
       unsigned int jj, kk, ll, mm;
       unsigned int num_bts;
       unsigned int increment;
       int *large_neg_integer_item;
       unsigned int *large_integer_item;
       unsigned short *medium_integer_item;
20       unsigned char *small_integer_item;
       unsigned int *bit_item;
       float *floating_item;
       double *double_item;
       char *string_value;
       char *string_ptr;
       unsigned int temp_counter;
       struct distribution_type temp_root_node;

       temp_root_node = *root_node;
       temp_counter = value_bitmap->start;
25       num_bts = value_bitmap->number_of_bts;

       kk = BITMAP_INTEGER_SIZE;
       jj = 1;
       current_pos = 0;

       for (i=0; i<num_bts; i++)
       {
30           if (!temp_counter)
           {
               if (temp_counter & jj)
               {
                   if (lookup_table->lookup_list[2] != NULL)
                   {
                       position = get_bucket_number(value[2], &lookup_table->lookup_list[2], 2, field_name[2],
35                           field_type[2], field_length[2], mcr, root_node);
                       row = get_bucket_number(value[1], &lookup_table->lookup_list[1], 1, field_name[1],
                           field_type[1], field_length[1], mcr, root_node);
                       column = get_bucket_number(value[0], &lookup_table->lookup_list[0], 0, field_name[0],
                           field_type[0], field_length[0], mcr, root_node);
                       temp_root_node->zposition[position]->yposition[row]->xposition[column]->total++;
                   }
                   else

```

```

    {
        position = 0;
        row = get_bucket_number(value[1], &lookup_table->lookup_list[1], 1, field_name[1],
                                field_type[1], field_length[1], incr, root_node);
        column = get_bucket_number(value[0], &lookup_table->lookup_list[0], 0, field_name[0],
                                field_type[0], field_length[0], incr, root_node);
        temp_root_node->xposition[position]->yposition[row]->xposition[column]->total++;
    }
    for (k=0; k<num_fields; k++)
    {
        increment = 1;
        if (sub_set[k])
            if (!temp_reference[k] & jj)
                temp_reference[k]++;
            else
                increment = 0;

        if (increment)
        {
            switch(field_type[k])
            {
                case dollars:
                case floating_point:
                case large_integer:
                case bit_type:
                    large_integer_item = (unsigned int *) value[k];
                    large_integer_item++;
                    value[k] = (void *) large_integer_item;
                    break;
                case negative_float:
                    large_neg_integer_item = (int *) value[k];
                    large_neg_integer_item++;
                    value[k] = (void *) large_neg_integer_item;
                    break;
                case fixed_string:
                case year_month:
                case year_month_day:
                case medium_integer:
                    medium_integer_item = (unsigned short *) value[k];
                    medium_integer_item++;
                    value[k] = (void *) medium_integer_item;
                    break;
                case character:
                case small_integer:
                    small_integer_item = (unsigned char *) value[k];
                    small_integer_item++;
                    value[k] = (void *) small_integer_item;
                    break;
                case string_type:
                    string_ptr = (unsigned char *) value[k];
                    string_ptr += field_length[k];
                    value[k] = (void *) string_ptr;
                    break;
            }
        }
        if (!kk)
            jj += 1;
        else
        {
            temp_counter++;
            jj = 1;
            kk = BITMAP_INTEGER_SIZE;
        }
    }
}

```



```

/* temp_counter == 0, bump past whole word */
else
{
    for (k=0; k<num_fields; k++)
    {
        if (sub_sel[k])
        {
            if (temp_reference[k])
            {
                for (ll=0; mm=1; ll<BITMAP_INTEGER_SIZE; ll++, mm<=1)
                {
                    if (temp_reference[k] & mm)
                    {
                        switch(field_type[k])
                        {
                            case dollars:
                            case floating_point:
                            case large_integer:
                            case bit_type:
                                large_integer_item = (unsigned int *) value[k];
                                large_integer_item++;
                                value[k] = (void *) large_integer_item;
                                break;
                            case negative_float:
                                large_neg_integer_item = (int *) value[k];
                                large_neg_integer_item++;
                                value[k] = (void *) large_neg_integer_item;
                                break;
                            case fixed_string:
                            case year_month:
                            case year_month_day:
                            case medium_integer:
                                medium_integer_item = (unsigned short *) value[k];
                                medium_integer_item++;
                                value[k] = (void *) medium_integer_item;
                                break;
                            case character:
                            case small_integer:
                                small_integer_item = (unsigned char *) value[k];
                                small_integer_item++;
                                value[k] = (void *) small_integer_item;
                                break;
                            case string_type:
                                string_ptr = (unsigned char *) value[k];
                                string_ptr += field_length[k];
                                value[k] = (void *) string_ptr;
                                break;
                        }
                    }
                }
                temp_reference[k]++;
            }
        }
        else
        {
            switch(field_type[k])
            {
                case dollars:
                case floating_point:
                case large_integer:
                case bit_type:
                    large_integer_item = (unsigned int *) value[k];
                    large_integer_item += BITMAP_INTEGER_SIZE;
                    value[k] = (void *) large_integer_item;
                    break;
                case negative_float:
                    large_neg_integer_item = (int *) value[k];
                    large_neg_integer_item += BITMAP_INTEGER_SIZE;
                    value[k] = (void *) large_neg_integer_item;
                    break;
                case fixed_string:
                case year_month:
            }
        }
    }
}

```

```

    case year_month_day:
    case medium_integer:
        medium_integer_rem = (unsigned short *) value[k];
        medium_integer_rem += BITMAP_INTEGER_SIZE;
        value[k] = (void *) medium_integer_rem;
        break;
    case character:
    case small_integer:
        small_integer_rem = (unsigned char *) value[k];
        small_integer_rem += BITMAP_INTEGER_SIZE;
        value[k] = (void *) small_integer_rem;
        break;
    case string_type:
        string_ptr = (unsigned char *) value[k];
        string_ptr += (field_length[k] * BITMAP_INTEGER_SIZE);
        value[k] = (void *) string_ptr;
        break;
    }
    i += BITMAP_INTEGER_SIZE - 1;
    temp_counter++;
}
return(*root_node);
}

/*
  Actually builds the distribution (X2 or X3) This routine will be called either for a purchase or product distribution
  The routine capture the row, column, and page position and updates the total for that array value
  */
20 struct distribution_type *show_mf_count_distribution(value.root_node, tail_node, field_name, field_type,
    field_precision, field_length, reference_map,
    num_fields, lookup_table, value_bitmap)
    void
    struct distribution_type *root_node,
    struct value_node *tail_node,
    char *field_name[],
    enum data_type field_type[],
    int field_precision[],
    unsigned int field_length[],
    unsigned short reference_map,
    int num_fields,
    struct lookup_table_list_type *lookup_table,
    struct bitmap *value_bitmap,
    (
        int row, column, position,
        unsigned int i, j, k, m;
        unsigned int inc=0;
        unsigned int jj, kk, ll, mm;
        unsigned int num_bns;
        int *large_neg_integer_rem;
        unsigned int *large_integer_rem;
        unsigned short *medium_integer_rem;
        unsigned char *small_integer_rem;
        unsigned int *int_rem;
        float *floating_rem;
        double *double_rem;
        char *string_value;
        char *string_ptr;
        unsigned int temp_counter;
        struct distribution_type temp_root_node;

```

```

temp_root_node = *root_node;
temp_counter = value_bitmap->start;
num_bits = value_bitmap->number_of_bits;

5  kk = BITMAP_INTEGER_SIZE;
   jj = 1;
   current_pos = 0;

   for (i=0; i<num_bits; i++)
   {
       if (*temp_counter)
       {
           if (*reference_map)
           {
10              if (*temp_counter & jj)
              {
                  for (incr = 0, incr < *reference_map; incr++)
                  {
                      if (lookup_table->lookup_list[2] != NULL)
                      {
15                          position = get_bucket_number(value[2], &lookup_table->lookup_list[2].field_name[2],
                                                              field_type[2], field_length[2], incr, root_node);
                          row = get_bucket_number(value[1], &lookup_table->lookup_list[1].field_name[1],
                                                              field_type[1], field_length[1], incr, root_node);
                          column = get_bucket_number(value[0], &lookup_table->lookup_list[0].field_name[0],
                                                              field_type[0], field_length[0], incr, root_node);
                          temp_root_node->zposition[position]->yposition[row]->xposition[column]->total++;
                      }
                      else
                      {
20                          position = 0;
                          row = get_bucket_number(value[1], &lookup_table->lookup_list[1].field_name[1],
                                                              field_type[1], field_length[1], incr, root_node);
                          column = get_bucket_number(value[0], &lookup_table->lookup_list[0].field_name[0],
                                                              field_type[0], field_length[0], incr, root_node);
                          temp_root_node->zposition[position]->yposition[row]->xposition[column]->total++;
                      }
                  }
              }
           }
       }

       for (k=0; k<num_fields; k++)
       {
25           switch(field_type[k])
           {
               case dollars:
               case floating_point:
               case large_integer:
               case bit_type:
                   large_integer_item = (unsigned int *) value[k];
                   large_integer_item += *reference_map;
                   value[k] = (void *) large_integer_item;
                   break;
30           case negative_float:
                   large_neg_integer_item = (int *) value[k];
                   large_neg_integer_item += *reference_map;
                   value[k] = (void *) large_neg_integer_item;
                   break;
               case fixed_string:
               case year_month:
               case year_month_day:
               case medium_integer:
35                   medium_integer_item = (unsigned short *) value[k];
                   medium_integer_item += *reference_map;
                   value[k] = (void *) medium_integer_item;
           }
       }
   }

```

```

                    break;
                case character:
                case small_integer:
                    small_integer_item = (unsigned char *) value[k];
                    small_integer_item += *reference_map;
                    value[k] = (void *) small_integer_item;
                    break;
                case string_type:
                    string_ptr = (unsigned char *) value[k];
                    string_ptr += (*reference_map * field_length[k]);
                    value[k] = (void *) string_ptr;
                    break;
            }
            reference_map++;
        }
        else
            reference_map++;

        if ( --kk )
            jj <= 1;
        else
        {
            temp_counter++;
            jj = 1;
            kk = BITMAP_INTEGER_SIZE;
        }
    }
    /* temp_counter == 0, bump past whole word */
    else
    {
        for (ii=0; ii < BITMAP_INTEGER_SIZE; ii++)
        {
            for (k=0; k<num_fields; k++)
            {
                switch(field_type[k])
                {
                    case dollars:
                    case floating_point:
                    case large_integer:
                    case bit_type:
                        large_integer_item = (unsigned int *) value[k];
                        large_integer_item += *reference_map;
                        value[k] = (void *) large_integer_item;
                        break;
                    case negative_float:
                        large_neg_integer_item = (int *) value[k];
                        large_neg_integer_item += *reference_map;
                        value[k] = (void *) large_neg_integer_item;
                        break;
                    case fixed_string:
                    case year_month:
                    case year_month_day:
                    case medium_integer:
                        medium_integer_item = (unsigned short *) value[k];
                        medium_integer_item += *reference_map;
                        value[k] = (void *) medium_integer_item;
                        break;
                    case character:
                    case small_integer:
                        small_integer_item = (unsigned char *) value[k];
                        small_integer_item += *reference_map;
                        value[k] = (void *) small_integer_item;
                        break;
                    case string_type:
                        string_ptr = (unsigned char *) value[k];
                        string_ptr += (field_length[k] * *reference_map);
                        value[k] = (void *) string_ptr;
                        break;
                }
            }
        }
    }
}

```

```

        )
        )
        reference_map++;
    }
    i += BITMAP_INTEGER_SIZE - 1;
    temp_counter++;
5    }
    }

    return(*root_node);
}

/*
   This routine is primarily used to count bits set for a single large integer.
   Not being current used.
10  */
int count_number_of_bits(brt_array)
int *brt_array;
{
    int num_count=0;

    union
    {
        int aa;
        unsigned char bb(4);
15    } val;

    if( val.aa == *brt_array )
    {
        num_count += bit_counts[val.bb(0)];
        num_count += bit_counts[val.bb(1)];
        num_count += bit_counts[val.bb(2)];
        num_count += bit_counts[val.bb(3)];
    }
20    return(num_count);
}

/*
   Distribution_type variable needs to put back all of the temporary buffers
   This routine frees the distribution structure handled by X2 or X3 distribution.
25  */
void free_distribution_type( distr_var, x_axis_count, y_axis_count, z_axis_count )
struct distribution_type *distr_var;
int x_axis_count;
int y_axis_count;
int z_axis_count;
{
    struct distribution_type temp;
    struct label_type label_var, label_var1;
    struct zposition zpos;
    struct yposition ypos;
    struct value_node xpos;
30    struct value_node vnptr, vnptr1;
    int i, position, row, column;

    temp = distr_var;

    for ( i=0; i<MAXFIELDS; i++ )
    {
        if ( (label_var = temp->fields[i]) != NULL )
        {
            while ( label_var != NULL )
35            {

```

176

```

        label_var1 = label_var->next;
        label_var->next = NULL;
#ifdef DBG
        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "FREE_DISTRIBUTION", "LABEL_VAL", label_var, "Freed.");
#endif
        free ( label_var );
        label_var = label_var1;
    }
    temp->fields[i] = NULL;
}
else
    break; /* done with label_type buffer pointers once first null is found */
}

for ( position=0; position < z_axis_count; position++ )
{
    if ( (zpos = temp->zposition[position]) != NULL )
    {
        for ( row=0; row < y_axis_count; row++ )
        {
            if ( (ypos = temp->zposition[position]->yposition[row]) != NULL )
            {
                for ( column=0; column < x_axis_count; column++ )
                {
                    if ( (xpos = temp->zposition[position]->yposition[row]->xposition[column]) != NULL )
                    {
                        vnptr = temp->zposition[position]->yposition[row]->xposition[column]->next_value;
#ifdef DBG
                        fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "FREE_DISTRIBUTION", "XPOS", xpos, "Freed.");
#endif
                        free ( xpos );
                        temp->zposition[position]->yposition[row]->xposition[column] = NULL;
                        if ( vnptr )
                        {
                            while ( vnptr != NULL )
                            {
                                vnptr1 = vnptr->next_value;
                                vnptr->next_value = NULL;
                                free ( vnptr );
                                vnptr = vnptr1;
                            }
                        }
                        temp->zposition[position]->yposition[row]->xposition[column]->next_value = NULL;
                    }
                }
            }
        }
    }
    else
        break;
}

#ifdef DBG
fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "FREE_DISTRIBUTION", "YPOS", ypos, "Freed.");
#endif
free ( ypos );
temp->zposition[position]->yposition[row] = NULL;
}
else
    break;
}

#ifdef DBG
fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "FREE_DISTRIBUTION", "ZPOS", zpos, "Freed.");
#endif
free ( zpos );
temp->zposition[position] = NULL;
}

#ifdef DBG
fprintf(debug_file, "%s\\%s\\%d\\%s\\n", "FREE_DISTRIBUTION", "TEMP", temp, "Freed.");
#endif
free ( temp );
temp = NULL;
}

```

```

/*
  This routine frees the lookup_table information handled by X2 or X3 distribution.
*/
void free_lookup_table_list_type ( lookup_dist )
struct lookup_table_list_type *lookup_dist;
{
5   struct lookup_table_type *lookup_ptr, *lookup_ptr_next;
   int i;

   for ( i=0; i<MAXFIELDS; i++ )
   {
       if ( (lookup_ptr=lookup_dist->lookup_list[i]) != NULL )
       {
           while ( lookup_ptr != NULL )
           {
10              lookup_ptr_next = lookup_ptr->next;
              lookup_ptr->next = NULL;

              #ifdef DBG
              fprintf(debug_file, "%s\n", "FREE_LOOKUP_TABLE_LIST_TYPE", "LOOKUP_PTR->VALUE",
                  lookup_ptr->value, "Freed.");
              #endif

              free ( lookup_ptr->value );
              lookup_ptr->value = NULL;

              #ifdef DBG
              fprintf(debug_file, "%s\n", "FREE_LOOKUP_TABLE_LIST_TYPE", "LOOKUP_PTR", lookup_ptr, "Freed.");
15              #endif

              free ( lookup_ptr );
              lookup_ptr = NULL;
              lookup_ptr = lookup_ptr_next;
           }
           lookup_dist->lookup_list[i] = NULL;
       }
       else
           break;
   }

20  #ifdef DBG
   fprintf(debug_file, "%s\n", "FREE_LOOKUP_TABLE_LIST_TYPE", "LOOKUP_DIST", lookup_dist, "Freed.");
   #endif
   free ( lookup_dist );
   lookup_dist = NULL;
}

/*
  This routine frees the lookup_table information handled by the DT, X2 or X3 distribution
*/
25 free_lookup_table_type ( lookup_ptr )
struct lookup_table_type *lookup_ptr;
{
   struct lookup_table_type *lookup_ptr_next;

   while ( lookup_ptr != NULL )
   {
       lookup_ptr_next = lookup_ptr->next;

       #ifdef DBG
       fprintf(debug_file, "%s\n", "FREE_LOOKUP_TABLE_TYPE", "LOOKUP_PTR->VALUE",
30          lookup_ptr->value, "Freed.");
       #endif

       free ( lookup_ptr->value );
       lookup_ptr->value = NULL;

       #ifdef DBG
       fprintf(debug_file, "%s\n", "FREE_LOOKUP_TABLE_TYPE", "LOOKUP_PTR", lookup_ptr, "Freed.");
       #endif

       free ( lookup_ptr );
       lookup_ptr = NULL;
       lookup_ptr = lookup_ptr_next;
35  }
}

```

```

/*
 * This routine frees the lookup_table information handled by DT distribution.
 */
free_lookup_table(lookup_ptr)
struct lookup_values *lookup_ptr;
{
    struct lookup_values *lookup_ptr_next;
5   while ( lookup_ptr != NULL )
    {
        lookup_ptr_next = lookup_ptr->next;
#ifdef DBG
        fprintf(debug_file, "%s\n", "FREE_LOOKUP_TABLE", "LOOKUP_PTR->LOOKUP_STRING",
            lookup_ptr->lookup_string);
        free ( lookup_ptr->lookup_string );
        lookup_ptr->lookup_string = NULL;
10    #endif
        free ( lookup_ptr );
        lookup_ptr = NULL;
        lookup_ptr = lookup_ptr_next;
    }
15 }

/*
 * This is the "X2 or X3" case. It is being called by the parser as "X2 or X3". It provides a sorted
 * distribution of a two fields (X2) or three fields (X3), using linked list, as opposed to a tree.
 */
int display_sample_cmt_distribution(file_list)
char *file_list;
{
    int status;
    int char(MAXFIELDS);
    int mod_buffer;
    char *file_ptr;
    char *table_name(MAXFIELDS);
    void *data(MAXFIELDS);
    enum data_type field_type(MAXFIELDS);
    int field_precision(MAXFIELDS);
    int field_length(MAXFIELDS);
    int field_number(MAXFIELDS);
    int sub_set(MAXFIELDS);
25   unsigned int temp_reference(MAXFIELDS);
    char field_name(MAXFIELDS)[FIELD_NAME_LENGTH+1];
    char *field_label(MAXFIELDS);
    char *filename;
    int subsidiary=0;
    struct address_range retaddr(MAXFIELDS);
    struct query_info *tempq;
    struct field_entry *field;
    struct field_entry *field_head=NULL;
    struct field_entry *field_tail=NULL;
30   int num_field=0;
    int x_axis_count=0;
    int y_axis_count=0;
    int z_axis_count=0;
    char delimiter=LOOKUP_DELIMITER;
    struct distribution_type *distribution=NULL;
    struct distribution_type *root_node = NULL;
    struct zposition_type *root_z_node = NULL;
    struct label_type *tail_label_node = NULL;
    struct value_node *tail_node = NULL;
35   char *buffer;
    int mf_case_flag = 0;
    int len;

```



```

char          msg_buffer[132];
char          *master_count_str,temp_count_str[NUM_STRING_SIZE];
struct query_info
int          *query_tree_head= NULL;
unsigned short reference_count;
5 struct bitmap reference_map;
struct lookup_values *reference_bitmap;
temp_lookup_buffer[MAXFIELDS];

/* initialize recurring values */
x_axis_length = 0;
y_axis_length = 0;
z_axis_length = 0;

max_row_value_count = 0;
max_col_value_count = 0;
10 max_page_value_count = 0;

lookup_table = NULL;
table_tail = NULL;

if (master_count == 0)
{
    strcpy(msg_buffer, "\n\nNOTE \n");
    strcat(msg_buffer, "The query you selected, could not match any records \n");
    15 strcat(msg_buffer, "No distribution was created. Please try another selection \n\n");
    if (DMO_CONNECTED)
        msg_buf_xfer(msg_buffer);
    else
        printf("%s", msg_buffer);

    return(SUCCESS);
}

20 /*
X2 or X3 can be done on multiple scenarios:
1. All customer fields - then get_master_bitmap will return the master_bitmap as the reference_bitmap.
2. Combination of customer and subsidiary fields - then get_master_bitmap will return the combination
of master_bitmap and the subsidiary_bitmap as the reference_bitmap. If SUB01 and SUB02 and CUS
are the fields, the reference_bitmap will be a combination of the master_bitmap, subsidiary_bitmap[1]
and subsidiary_bitmap[2] and so on
3. All purchase fields - if (purchase_query) was set then get_master_bitmap will return the master_purchase
bitmap as the reference_bitmap. Else, the master_bitmap is returned and the mf_case_flag is then set
25 4. All product fields - if (product_query) was set then get_master_bitmap will return the master_product
bitmap as the reference_bitmap. Else, the master_bitmap is returned and the mf_case_flag is then set
*/

reference_bitmap = get_master_bitmap(&mf_case_flag);

if (reference_bitmap == NULL)
{
    30 strcpy(msg_buffer, "\n\nNOTE \n");
    strcat(msg_buffer, "An invalid combination of distribution field(s) selected \n");
    strcat(msg_buffer, "Please submit an SPR with the Database Link Product Manager.\n\n");
    if (DMO_CONNECTED)
        msg_buf_xfer(msg_buffer);
    else
        printf("%s", msg_buffer);

    return(SUCCESS);
}

35

```

```

    )
    if ((mf_case_flag) &&
        (purchase_query || product_query))
5   {
        sprintf(temp_count_str, "%d", master_count);
        master_count_str = insert_comma(temp_count_str);
        sprintf(msg_buffer, "Total customer records      : %s\n", master_count_str);

        if (DMO_CONNECTED)
            msg_buf_xfer(msg_buffer);
        else
            printf("%s\n", msg_buffer);

10     free(master_count_str);
        master_count_str = NULL;
    }

    /* initialize variables */
    for (i=0; i<MAXFIELDS; i++)
    {
        sub_set[i] = 0;
        temp_reference[i] = 0;
15    }

    if (strcmp(file_list, "")==0)
    {
        /* file_list was not specified */
        printf("Enter either the filename or distribution frequency: ");
        scanf("%s", file_list);
    }

    i = 0;
20    mod_buffer = file_list;
    while ((file_ptr=strchr(mod_buffer, delimiter)) != NULL)
    {
        filename = get_string(&mod_buffer, delimiter);
        table_name[i] = malloc(MAXFIELDS+1);
        CHECK_ALLOCATION(table_name[i], "table_name[i] display_sample_cnt_distribution()");

        if ((file_ptr=strchr(filename, '.')) != NULL)
        {
25             sprintf(table_name[i], "%s", filename);
        }
        else
        {
            sprintf(table_name[i], "lookup_dir_%s.dat", filename);
        }
        table_name[i][strlen(table_name[i])] = '\0';
        free(filename);
        filename=NULL;
30    }

    if (file_ptr==NULL)
    {
        filename = get_string(&mod_buffer, delimiter);
        table_name[i] = malloc(MAXFIELDS+1);
        CHECK_ALLOCATION(table_name[i], "table_name[i] display_sample_cnt_distribution()");

        if ((file_ptr=strchr(filename, '.')) != NULL)
        {
35             sprintf(table_name[i], "%s", filename);
        }
        else
        {
            sprintf(table_name[i], "lookup_dir_%s.dat", filename);
        }
        table_name[i][strlen(table_name[i])] = '\0';
    }

```

```

    free(filename);
    filename = NULL;
}

5  i=0;
   while((tempq = next_parse_entry()) != NULL)
   {
       if (query_tree_head == NULL)
           query_tree_head = tempq;
       else
       {
           free_parse_tree(query_tree_head);
           query_tree_head = tempq;
10      }

       field = tempq->field;
       strcpy(field_name[num_field], tempq->field->field_name);
       field_number[num_field] = hash(tempq->field->field_name);
       field_label[num_field] = get_field_label(tempq->field->field_name, tempq->field->field_label);

       field_type[num_field] = tempq->field->field_type;
       field_precision[num_field] = tempq->field->field_precision;
       field_length[num_field] = tempq->field->field_end;

15      if (strcmp(field->field_name, "SUB") == 0)
      {
          sub_set[num_field] = 1;
          subsidiary = 1;
          sscanf(tempq->field->field_name+3, "%d", &reference_count);
          temp_reference[num_field] = subsidiary_bitmap(reference_count)->start;
      }

20      /* map section file */
      status = OpenMapFile(tempq->field->table, "toolbar.sec", tempq->field->field_name,
                          &char[num_field], &retadr[num_field], tempq->field->vbn,
                          tempq->field->number_of_blocks);

      if (is_error(status))
          error_handler (MAP_OPEN_ERR, ERROR, status, "display_sample_cnt_distribution");

      /* set start address of data points */
25      data[num_field] = retadr[num_field].start;
      data[num_field+1] = NULL;
      tempq->field->data->nems = retadr[num_field].start;

      if (y_axis_count <= 0)
          y_axis_count = num_field - 1;
      if (z_axis_count <= 0)
          z_axis_count = num_field - 2;

      lookup_buffer = lookup_val_tail = NULL;

30      load_sample_count_header(&root_node, &root_z_node, &tail_label_node, table_name[num_field],
                              data[num_field], field_name[num_field], field_type[num_field], field_precision[num_field],
                              field_length[num_field], sub_set[num_field], temp_reference[num_field],
                              ml_case_flag, num_field,
                              &x_axis_count, &y_axis_count, &z_axis_count, reference_bitmap);

      temp_lookup_buffer[num_field] = lookup_buffer;
      num_field++;
   }

35  if (num_field == 2)
  {
      /* no z field required for displaying, assume z-axis is zero */
      root_node->zposition[0] = root_z_node;
      z_axis_count = 1;
  }

```

```

/* search for value distribution */
if ( mf_case_flag )
{
    if ( strcmp(field->field_name, "PUR", 3) == 0)
    reference_map = pur01_map;
5      else
        reference_map = prd01_map;
        distribution = show_mf_count_distribution(data, &root_node, &tail_node, field_name,
                                                field_type, field_precision, field_length, reference_map,
                                                num_field, lookup_table, reference_bitmap);
    }
    else
    {
        distribution = show_sample_count_distribution(data, &root_node, &tail_node, field_name,
10      field_type, field_precision, field_length, sub_sel, temp_reference,
        num_field, lookup_table, reference_bitmap);
    }

    free_bitmap(reference_bitmap);
    reference_bitmap = NULL;

    for (i=0; i<num_field; i++)
    {
        /* free the table name structure */
        free(table_name[i]);
        table_name[i] = NULL;

        /* unmap section file */
        status = UnmapCloseFile(chan[i], &reladr[i]);
        if (is_error(status))
15      error_handler (UNMAP_CLOSE_ERR, ERROR, status, "display_sample_cnt_distribution")
    }

    /* print value distribution */
    if (distribution != NULL)
20    {
        /* svp - we will be using the same print calls for crosstab and rlm */
        /* since we will be printing count values */
        /* print_3distribution should be called if you want to print the */
        /* actual dollar values as opposed to the total count */

        if (num_field == 3)
            print_3distribution(distribution, field_label, x_axis_count, y_axis_count, z_axis_count);
        else
25      print_2distribution(distribution, field_label, x_axis_count, y_axis_count, z_axis_count, x_axis_length,
            y_axis_length, z_axis_length);
    }
    else
    {
        sprintf(msg_buffer, "\n\nNOTE.\n");
        strcat(msg_buffer, "No values found for this distribution, please try another query.\n\n");
        if (DMQ_CONNECTED)
            msg_buf_xfer(msg_buffer);
        else
30      printf("%s\n", msg_buffer);
        print_text_bufs( );
    }

    /* initialize values */
    create_z_table = FALSE;
    create_y_table = FALSE;
    create_x_table = FALSE;
35

    /*
    Although distribution fields are build within the query_info structure, they are not queries, so
    there is not history associated with that field and in addition the current_query_entry needs to
    be decremented to compensate for this.
    */
    reset_query_entry(num_field);

```

```

    if (query_tree_head != NULL)
    {
        free_parse_tree (query_tree_head);
        query_tree_head = NULL;
        reset_parse_tree();
    }

    /* need to put back all of the malloc'd buffers in this compound structure
    /* note: distribution and root_node are exactly the same here. put back only once
    if (distribution != NULL)
    {
        free_distribution_type( distribution, x_axis_count, y_axis_count, z_axis_count );
        distribution = NULL;
    }

    /* free all the parts to lookup_distr_table
    if (lookup_table != NULL)
    {
        free_lookup_table_list_type ( lookup_table );
        table_tail = NULL;
        lookup_table = NULL;
    }

    /* free the label information */
    for (i=0; i<num_field; i++)
    {
        free(field_label[i]);
        field_label[i] = NULL;

        /* free the lookup_buffer */
        if (temp_lookup_buffer[i] != NULL)
        {
            free_lookup_table(temp_lookup_buffer[i]);
            temp_lookup_buffer[i] = NULL;
        }
    }

    return(SUCCESS);
}

#ifdef OLD_FUNC
/*
    This routine is not being used currently
    For every bit set in the customer bitmap (master_bitmap), the appropriate field value is stored
    The stored value can then be printed or written to a file
    */
struct value_node *show_value_distribution(data, field_type, field_length, field_offset, value_bitmap)
struct dataset *data;
enum data_type field_type;
unsigned int field_length;
int field_offset;
struct bitmap *value_bitmap;
{
    unsigned int i, j, jj, kk;
    unsigned int num_bits;
    unsigned long *long_rem;
    unsigned int *large_integer_rem;
    unsigned short *medium_integer_rem;
    unsigned char *small_integer_rem;
    unsigned int *bit_rem;
    unsigned int temp_counter;
    float *floating_rem;
    double *double_rem;
    char *string_value;
    char *string_ptr;
}

```

```

    struct value_node *root_node = NULL;
    struct value_node *tail_node = NULL;

    string_value = malloc(field_length+1);
5    CHECK_ALLOCATION(string_value, "string_value:show_value_distribution()");

    temp_counter = value_bitmap->start;

    num_bits = data->number_of_items;

    switch (field_type)
    {
        case dollars:
        case floating_point:
        case large_integer:
            PROCESS_BIT_TABLE_PART_1( large_integer_item, unsigned int * );
            insert_value( large_integer_item, &root_node, &tail_node, field_type, field_length);
            PROCESS_BIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
            break;
        case fixed_string:
        case year_month:
        case year_month_day:
        case medium_integer:
15        PROCESS_BIT_TABLE_PART_1( medium_integer_item, unsigned short * );
            insert_value( medium_integer_item, &root_node, &tail_node, field_type, field_length);
            PROCESS_BIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
            break;
        case character:
        case small_integer:
            PROCESS_BIT_TABLE_PART_1( small_integer_item, unsigned char * );
            insert_value( small_integer_item, &root_node, &tail_node, field_type, field_length);
            PROCESS_BIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
            break;
20        case string_type:
            PROCESS_BIT_TABLE_PART_1( string_ptr, unsigned char * );
            strcpy(string_value, string_ptr, field_length);
            string_value[field_length] = '\0';
            insert_value(string_value, &root_node, &tail_node, field_type, field_length);
            PROCESS_BIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE, field_length, field_length);
            break;
        case bit_type:
25        PROCESS_BIT_TABLE_PART_1( bit_item, unsigned int * );
            insert_value( (bit_item & 1), &root_node, &tail_node, field_type, field_length);
            PROCESS_BIT_TABLE_PART_2( bit_item, BITMAP_INTEGER_SIZE, 1 );
            break;
    }

    #ifdef DBG
        fprintf(debug_file, "%s\n", "DISPLAY_SAMPLE_CNT_DISTRIBUTION", "STRING_VALUE",
            string_value, "Freed.");
    #endif
30    free( string_value );
    string_value = NULL;

    return(root_node);
}
#endif

```

35

```

#endif OLD_FUNC
/*
  THIS ROUTINES HAVE BEEN OUT OF SYNC WITH THE DISTRIBUTION. PLEASE CHECK THE VALIDITY
  */
/*
  This routine is not being used currently
  This routine will display a field value on screen.
  */
5 int show_field_distribution(field)
  struct field_entry *field;
{
    int status;
    int chan;
    struct address_range *retadr;
    struct value_node *distribution;
    char *buffer;
    10 char msg_buffer[132];
    int field_offset = 0;

    if (master_count == 0)
    {
        strcpy(msg_buffer, "Query did not produce any results.\n");
        strcat(msg_buffer, "No distribution created.\n\n");
        if (DMO_CONNECTED)
            msg_buf_xfer(msg_buffer);
        15 else
            printf("%s", msg_buffer);
        return(SUCCESS);
    }

    /* map section file */
    status = OpenMapFile(field->table "foobar sec", field->field_name &chan,
        &retadr, field->vbn, field->number_of_blocks);
    20 if (is_error(status))
        error_handler(MAP_OPEN_ERR, ERROR, status, "show_field_distribution");

    /* set start address of data points */
    field->data->items = retadr->start;

    if (field->field_type == fixed_string)
        field_offset = get_fixed_string_offset(field->field_name);

    /* search for value distribution */
    25 distribution = show_value_distribution(field->data, field->field_type,
        field->field_end, field_offset, master_bitmap);

    /* print value distribution */
    buffer = malloc(STR_BUFFER_LENGTH);
    CHECK_ALLOCATION(buffer, "buffer Routine show_field_distribution()");
    sprintf(buffer, "Value distribution for %s\n", field->field_name);
    msg_buf_xfer(buffer);

    30 if (distribution != NULL)
        /* print_value_tree will free all the various nodes in distribution */
        print_value_tree(distribution, field->field_type, field->field_end);
    else
    {
        sprintf(buffer, "No values found for this distribution, please try another query\n");
        msg_buf_xfer(buffer);
        print_text_buf(buffer);
    }

    35 free(buffer);
    buffer = NULL;

    /* unmap section file */
    status = UnmapCloseFile(chan, &retadr);
    if (is_error(status))
        error_handler(UNMAP_CLOSE_ERR, ERROR, status, "show_field_distribution");

    return(SUCCESS);
}
#endif

```

```

#define OLD_FUNC
/*
THIS ROUTINES HAVE BEEN OUT OF SYNC WITH THE DISTRIBUTION. PLEASE CHECK THE VALIDITY.
*/
/*
This routine is not being used currently.
For every bit set in the customer bitmap (master_bitmap), the appropriate field value is stored.
The stored value can then be printed or written to a file.
Since this routine can be used to store multiple field values, unlike a single field value in show_value_distribution,
it is generally preferable to use this routine to display a number of field values on screen. However, the entire field
values are initially stored in a linked list prior to being either written or printed.
*/
5 struct list_value_node *show_list_value_distribution(data, root_node, tail_node, field_number, field_type, field_length,
  f_offset, value_bitmap)
  struct dataset *data;
  struct list_value_node *root_node;
  struct value_node *tail_node;
  unsigned int field_number;
  enum data_type field_type;
  unsigned int field_length;
  int f_offset;
  struct bitmap *value_bitmap;
  {
    unsigned int i, j, k;
    unsigned int num_bits;
    unsigned long *long_item;
    unsigned int *large_integer_item;
    unsigned short *medium_integer_item;
    unsigned char *small_integer_item;
    unsigned int *bit_item;
    unsigned int temp_counter;
    float *floating_item;
    double *double_item;
    char *string_value;
    char *string_ptr;

    15 string_value = malloc(field_length+1);
    CHECK_ALLOCATION(string_value, "string_value: Routine show_list_value_distribution()");

    temp_counter = value_bitmap->start;

    num_bits = data->number_of_items

    switch (field_type)
    {
      case dollars:
      case floating_point:
      case large_integer:
        25 PROCESS_BIT_TABLE_PART_1( large_integer_item, unsigned int * ),
            insert_list_value( large_integer_item, root_node, tail_node, field_type,
                                field_number, field_length);
        PROCESS_BIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
      case year_month:
      case year_month_day:
      case medium_integer:
        30 PROCESS_BIT_TABLE_PART_1( medium_integer_item, unsigned short * ),
            insert_list_value( medium_integer_item, root_node, tail_node,
                                field_type, field_number, field_length);
        PROCESS_BIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
      case character:
      case small_integer:
        PROCESS_BIT_TABLE_PART_1( small_integer_item, unsigned char * );
            insert_list_value( small_integer_item, root_node, tail_node, field_type,
                                field_number, field_length);
        35 PROCESS_BIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
      case string_type:
        PROCESS_BIT_TABLE_PART_1( string_ptr, unsigned char * );
            strcpy(string_value, string_ptr, field_length);
            string_value[field_length] = '\0';
    }
  }

```



```

        insert_list_value(string_value, root_node, tail_node, field_type, field_number, field_length);
        PROCESS_BIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE, field_length, field_length );
        break;
    case bit_type
        PROCESS_BIT_TABLE_PART_1( bit_item, unsigned int * );
        insert_list_value((bit_item & j), root_node, tail_node, field_type, field_number, field_length);
5      PROCESS_BIT_TABLE_PART_2( bit_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case fixed_string
        PROCESS_BIT_TABLE_PART_1( medium_integer_item, unsigned short * );
        strncpy(string_value, fixed_field[i_offset], fixed_string[medium_integer_item]);
        string[field_length] = '\0';
        insert_list_value(string_value, root_node, tail_node, field_type, field_number, field_length);
        PROCESS_BIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
10  }

    free( string_value );
    string_value = NULL;

    return( root_node );
}
#endif

15  #endif OLD_FUNC

    THIS ROUTINES HAVE BEEN OUT OF SYNC WITH THE DISTRIBUTION. PLEASE CHECK THE VALIDITY.
    /*
    /*
    /* This routine is not being used currently
    /* This routine displays multiple field values on screen. However, the values are stored in a linked
    /* list prior to being printed
    /*
20  int show_list_field_distribution()
    {
        int status;
        int chan;
        struct address_range reladdr;
        struct query_info tempq;
        struct field_entry field;
        unsigned int field_number;
        int num_fields=0;
25  struct list_value_node *distribution;
        struct list_value_node *root_node = NULL;
        struct value_node tail_node = NULL;
        FILE *f;
        char *buffer;
        char msg_buffer[132];
        int field_offset=0;
        struct query_info *query_tree_head = NULL;
30  if (master_count == 0)
    {
        strcpy(msg_buffer, "Query did not produce any results.\n");
        strcat(msg_buffer, "No distribution created v\n");
        if (DMQ_CONNECTED)
            msg_buf_xfer(msg_buffer);
        else
            printf("%s", msg_buffer);

        return(SUCCESS);
35  }

    while((tempq = next_parse_entry()) != NULL)
    {
        if (query_tree_head == NULL)
            query_tree_head = tempq;
        else
        {

```

```

        free_parse_tree(query_tree_head);
        query_tree_head = tempq;
    }

    field = tempq->field;
    /* map section file */
    status = OpenMapFile(field->table, "foobar.sec", field->field_name
                        &chan, &retadr, field->von, field->number_of_blocks);
    if (is_error(status))
        error_handler(MAP_OPEN_ERR, ERROR, status, "show_list_field_distribution");

    /* set start address of data points */

    field->data->items = retadr.start;
    field_number = hash(field->field_name);

    if (field->field_type == fixed_string)
        field_offset = get_fixed_string_offset(field->field_name);

    /* search for value distribution */
    distribution = show_list_value_distribution(field->data, &root_node,
                                              &tail_node, field_number, field->field_type,
                                              field->field_end, field_offset, master_bitmap);

    /* unmap section file */
    status = UnmapCloseFile(chan, &retadr);
    if (is_error(status))
        error_handler(UNMAP_CLOSE_ERR, ERROR, status, "show_list_field_distribution");

    num_fields++;
}

/* print value distribution */
/* DMO_CONNECTED case always uses file case and print_file_value_tree */
if (DMO_CONNECTED)
{
    /* the filename is built in this manner */
    sprintf(msg_file_buffer, "write_dir %s", query_filename);

    if ((f = fopen(msg_file_buffer, "a+")) != NULL)
    {
        fprintf(f, "%s-%s\n", query_keycode, query_label);
        if (distribution != NULL)
        {
            /* print_file_value_tree will free all the nodes in distribution */
            print_file_value_tree(f, distribution, num_fields, field->field_type);
        }
        else
            fprintf(f, "No values found for this distribution, please try another query.\n");
    }
    else
    {
        error_handler(FILE_NOT_OPEN, ERROR, NO_STATUS, msg_file_buffer);
    }
}
if (fclose(f) == EOF)
{
    error_handler(FILE_NOT_CLOSE, ERROR, NO_STATUS, msg_file_buffer);
}
else
{
    /* if not DMO_CONNECTED, doing local show with direct printf(s) using print_list_value_tree */
    if (distribution != NULL)
    {
        /* print_list_value_tree will free all the nodes in distribution */
        print_list_value_tree(distribution, num_fields, field->field_type);
    }
    else
        printf("No values found for this distribution, please try another query.\n");
}
}

```

```

    /
    add_parse_history().
    /
    if (query_tree_head != NULL)
    {
        free_parse_tree (query_tree_head);
        free(query_tree_head);
    }

    return(SUCCESS);
}
#endif

#ifndef OLD_FUNC
/
10  THIS ROUTINES HAVE BEEN OUT OF SYNC WITH THE DISTRIBUTION. PLEASE CHECK THE VALIDITY
    /
    This routine is not being used currently
    This routine writes multiple field values on to a file. However,
    the values are stored in a linked list prior to being written.
    /
    int write_field_distribution()
    {
15      int                status;
        int                chan;
        struct address_range retaddr;
        struct query_info *tempq;
        struct field_entry *field;
        unsigned int      field_number;
        int                num_fields=0;
        struct list_value_node *distribution;
        struct list_value_node *root_node = NULL;
        struct value_node *tail_node = NULL;
20      FILE                *f;
        int                field_offset=0;
        char                *buffer;
        char                msg_buffer[132];
        struct query_info    *query_tree_head= NULL;

        if (master_count == 0)
        {
            strcpy(msg_buffer, "Query did not produce any results\n");
            strcat(msg_buffer, "No distribution created\n");
25          if (DMQ_CONNECTED)
                msg_buf_xfer(msg_buffer);
            else
                printf("%s", msg_buffer);

            return(SUCCESS);
        }

        / the filename is built in this manner */
30      sprintf(msg_file_buffer, "write_dir%s", query_filename);

        if ((f = fopen(msg_file_buffer, "a+")) != NULL)
        {
            while((tempq = next_parse_entry()) != NULL)
            {
                if (query_tree_head == NULL)
                    query_tree_head = tempq;
                else
35                {
                    free_parse_tree (query_tree_head);
                    query_tree_head = tempq;
                }
            }
        }
    }

```

```

    }

    field = tempq->field;
    /* map section file */
    status = OpenMapFile(field->table, "toolbar.sec", field->field_name,
                        &chan, &retadr, field->von, field->number_of_blocks);
5    if (is_error(status))
        error_handler (MAP_OPEN_ERR, ERROR, status, "write_field_distribution");

    /* set start address of data points */
    field->data->nems = retadr start;
    field_number = hash(field->field_name);

    if (field->field_type == fixed_string)
        field_offset = get_fixed_string_offset(field->field_name);
10    /* search for value distribution */
    distribution = show_list_value_distribution(field->data, &root_node,
        &tail_node, field_number, field->field_type,
        field->field_end, field_offset, master_bitmap);

    /* unmap section file */
    status = UnmapCloseFile(chan, &retadr);
    if (is_error(status))
        error_handler (UNMAP_CLOSE_ERR, ERROR, status, "write_field_distribution");
15    num_fields++
}

/* print value distribution */
fprintf(ff, "%s-%s\n", query_keycode, query_label);

if (distribution != NULL)
    /* print_file_value_tree will free all the nodes in distribution */
    print_file_value_tree(ff, distribution, num_fields, field->field_type);
20    else
    {
        fprintf(ff, "No values found for this distribution, please try another query\n");
    }
}
else
{
    error_handler(FILE_NOT_OPEN, ERROR, NO_STATUS, msg_file_buffer);
25    if (fclose(ff) == EOF)
        error_handler (FILE_NOT_CLOSE, ERROR, NO_STATUS, msg_file_buffer);
}

30    /*
    add_parse_history();
    */

    if (query_tree_head != NULL)
    {
        free_parse_tree (query_tree_head);
        free(query_tree_head);
    }
35    return(SUCCESS);
}
#endif

```

```

/*
  For every bit set in the customer bitmap (master_bitmap), the appropriate field value is written to a file
*/
int write_value_distribution(file_id, file_location, data, field_type, field_length, field_offset, value_bitmap)
int      file_id;
int      file_location;
5 struct dataset      *data;
enum data_type      field_type;
unsigned int      field_length;
int      field_offset;
struct bitmap      *value_bitmap;
{
    unsigned int      i, j, jj, kk;
    unsigned int      num_bits;
10 unsigned long      *long_item;
    int      *large_neg_integer_item, neg_integer_value;
    unsigned int      *large_integer_item, integer_value;
    unsigned short      *medium_integer_item, medium_integer_value;
    unsigned char      *small_integer_item;
    unsigned int      *bit_item;
    unsigned int      temp_counter;
    float      *floating_item, float_value;
    double      *double_item, double_value;
15 char      *string_value;
    char      *string_ptr;
    char      fixed_string_value[FIXED_STRING_LENGTH];
    int      current_position=0;

    struct value_node *root_node = NULL;
    struct value_node *tail_node = NULL;

    string_value = malloc(field_length+1);
    CHECK_ALLOCATION(string_value "string_value.show_value_distribution()");
20 temp_counter = value_bitmap->start;

    num_bits = data->number_of_items;

    switch (field_type)
    {
        case character
            PROCESS_BIT_TABLE_PART_1( small_integer_item, unsigned char * ),
25 lseek(file_id, file_location, 0);
            sprintf(fixed_string_value "%c", (int *)small_integer_item);
            write(file_id, fixed_string_value, sizeof(int));
            file_location += PAGE_SIZE;
            current_position = sizeof(int);
            PROCESS_BIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
            break;
        case small_integer
            PROCESS_BIT_TABLE_PART_1( small_integer_item, unsigned char * ),
30 lseek(file_id, file_location, 0);
            sprintf(fixed_string_value "%4d", (char *)small_integer_item);
            write(file_id, (char *)fixed_string_value, sizeof(int));
            file_location += PAGE_SIZE;
            current_position = sizeof(int);
            PROCESS_BIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
            break;
        case bit_type
        case large_integer
            PROCESS_BIT_TABLE_PART_1( large_integer_item, unsigned int * );
35 lseek(file_id, file_location, 0);
            sprintf(fixed_string_value "%10d", (int *)large_integer_item);
            write(file_id, (char *)fixed_string_value, field_length);
            file_location += PAGE_SIZE;
            current_position = sizeof(int);
    }
}

```

```

PROCESS_BIT_TABLE_PART_2( large_integer_rem, BITMAP_INTEGER_SIZE, 1 );
break;
case dollars
PROCESS_BIT_TABLE_PART_1( large_integer_rem, unsigned int * ),
5      lseek(file_id, file_location, 0);
      integer_value = (int *)large_integer_rem;
      double_value = (double)integer_value/100.0;
      sprintf(fixed_string_value, "%7.2f", (double)double_value);
      write(file_id, (char *)fixed_string_value, sizeof(int));
      file_location += PAGE_SIZE;
      current_position = sizeof(double);
PROCESS_BIT_TABLE_PART_2( large_integer_rem, BITMAP_INTEGER_SIZE, 1 );
break;
10 case floating_point:
PROCESS_BIT_TABLE_PART_1( large_integer_rem, unsigned int * ),
      lseek(file_id, file_location, 0);
      integer_value = (int *)large_integer_rem;
      float_value = (float)integer_value/100.0;
      sprintf(fixed_string_value, "%7.2f", (float)float_value);
      write(file_id, (char *)fixed_string_value, sizeof(int));
      file_location += PAGE_SIZE;
      current_position = sizeof(float);
PROCESS_BIT_TABLE_PART_2( large_integer_rem, BITMAP_INTEGER_SIZE, 1 );
break;
15 case negative_float:
PROCESS_BIT_TABLE_PART_1( large_neg_integer_rem, int * ),
      lseek(file_id, file_location, 0);
      neg_integer_value = (int *)large_neg_integer_rem;
      float_value = (float)neg_integer_value/100.0;
      sprintf(fixed_string_value, "%7.2f", (float)float_value);
      write(file_id, (char *)fixed_string_value, sizeof(int));
      file_location += PAGE_SIZE;
      current_position = sizeof(float);
PROCESS_BIT_TABLE_PART_2( large_neg_integer_rem, BITMAP_INTEGER_SIZE, 1 );
20 break;
case medium_integer:
PROCESS_BIT_TABLE_PART_1( medium_integer_rem, unsigned short * ),
      lseek(file_id, file_location, 0);
      sprintf(fixed_string_value, "%4d", (short *)medium_integer_rem);
      write(file_id, (char *)fixed_string_value, sizeof(int));
      file_location += PAGE_SIZE;
      current_position = sizeof(int);
PROCESS_BIT_TABLE_PART_2( medium_integer_rem, BITMAP_INTEGER_SIZE, 1 );
25 break;
case fixed_string:
PROCESS_BIT_TABLE_PART_1( medium_integer_rem, unsigned short * ),
      lseek(file_id, file_location, 0);
      string_value = malloc(field_length+1);
      CHECK_ALLOCATION(string_value, "string_value Routine write_value_distribution()");
      strcpy(string_value, fixed_string_value);
      string_value[field_length] = '\0';
      write(file_id, string_value, field_length);
30 free ( string_value );
      string_value = NULL;
      file_location += PAGE_SIZE;
      current_position = field_length;
PROCESS_BIT_TABLE_PART_2( medium_integer_rem, BITMAP_INTEGER_SIZE, 1 );
break;
case year_month:
case year_month_day:
35 PROCESS_BIT_TABLE_PART_1( medium_integer_rem, unsigned short * );
      lseek(file_id, file_location, 0);
      sprintf(fixed_string_value, "%s", number_to_date("medium_integer_rem"));
      write(file_id, (char *)fixed_string_value, 10);
      file_location += PAGE_SIZE;
      current_position = 10;

```

```

PROCESS_BIT_TABLE_PART_2( medium_integer_rem, BITMAP_INTEGER_SIZE, 1 ).
break;
case string_type:
PROCESS_BIT_TABLE_PART_1( string_ptr, unsigned char * ).
5       lseek(file_id, file_location, 0);
       string_value = malloc(field_length+1);
       CHECK_ALLOCATION(string_value, "string_value: Routine write_value_distribution()")
       strncpy(string_value, (char *)string_ptr, field_length);
       string_value[field_length] = '\0';
       write(file_id, string_value, field_length);

       free ( string_value );
       string_value = NULL;
       file_location += PAGE_SIZE;
       current_position = field_length;
10      PROCESS_BIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE*field_length, field_length )
       break;
    }

return(current_position);
}

/*
For every bit set in the customer bitmap (master_bitmap), the appropriate field value is written to a file.
*/
15 int write_subs_value_distribution(file_id, file_location, data, field_type, field_length, field_offset, value_bitmap,
reference_bitmap)
int file_id;
int file_location;
struct dataset *data;
enum data_type field_type;
unsigned int field_length;
int field_offset;
struct bitmap *value_bitmap;
struct bitmap *reference_bitmap;
20 {
    unsigned int i, j, k, l, m;
    unsigned int num_bits;
    unsigned long *long_rem;
    int *large_neg_integer_rem, neg_integer_value;
    unsigned int *large_integer_rem, integer_value;
    unsigned short *medium_integer_rem, medium_integer_value;
    unsigned char *small_integer_rem;
    unsigned int *bit_rem;
    unsigned int temp_counter, temp_reference;
    float *floating_rem, float_value;
    double *double_rem, double_value;
    char *string_value;
    char *string_ptr;
    char fixed_string_value(FIXED_STRING_LENGTH);
    int current_position=0;

    struct value_node *root_node = NULL;
    struct value_node *tail_node = NULL;
    30 string_value = malloc(field_length+1);
    CHECK_ALLOCATION(string_value, "string_value: show_value_distribution()").

    temp_counter = value_bitmap->start;
    temp_reference = reference_bitmap->start;

    num_bits = value_bitmap->number_of_bits;

    35 switch (field_type)
    {

```

```

case character
    PROCESS_MBIT_TABLE_PART_1( small_integer_item, unsigned char * ),
        lseek(file_id, file_location, 0),
        sprintf(fixed_string_value, "%c", (int *)small_integer_item),
        write(file_id, fixed_string_value, sizeof(int)),
        file_location += PAGE_SIZE,
        current_position = sizeof(int);
    PROCESS_MBIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
break;

case small_integer
    PROCESS_MBIT_TABLE_PART_1( small_integer_item, unsigned char * ),
        lseek(file_id, file_location, 0),
        sprintf(fixed_string_value, "%4d", (char *)small_integer_item),
        write(file_id, (char *)fixed_string_value, sizeof(int)),
        file_location += PAGE_SIZE,
        current_position = sizeof(int);
    PROCESS_MBIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
break;

case bit_type
case large_integer
    PROCESS_MBIT_TABLE_PART_1( large_integer_item, unsigned int * ),
        lseek(file_id, file_location, 0),
        sprintf(fixed_string_value, "%10d", (int *)large_integer_item),
        write(file_id, (char *)fixed_string_value, field_length),
        file_location += PAGE_SIZE,
        current_position = sizeof(int);
    PROCESS_MBIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
break;

case dollars
    PROCESS_MBIT_TABLE_PART_1( large_integer_item, unsigned int * ),
        lseek(file_id, file_location, 0),
        integer_value = (int *)large_integer_item;
        double_value = (double)integer_value/100.0;
        sprintf(fixed_string_value, "%7.2f", (double)double_value);
        write(file_id, (char *)fixed_string_value, sizeof(int)),
        file_location += PAGE_SIZE,
        current_position = sizeof(double);
    PROCESS_MBIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
break;

case floating_point
    PROCESS_MBIT_TABLE_PART_1( large_integer_item, unsigned int * ),
        lseek(file_id, file_location, 0),
        integer_value = (int *)large_integer_item;
        float_value = (float)integer_value/100.0;
        sprintf(fixed_string_value, "%7.2f", (float)float_value);
        write(file_id, (char *)fixed_string_value, sizeof(int)),
        file_location += PAGE_SIZE,
        current_position = sizeof(float);
    PROCESS_MBIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
break;

case negative_float
    PROCESS_MBIT_TABLE_PART_1( large_neg_integer_item, int * ),
        lseek(file_id, file_location, 0),
        neg_integer_value = (int *)large_neg_integer_item;
        float_value = (float)neg_integer_value/100.0;
        sprintf(fixed_string_value, "%7.2f", (float)float_value);
        write(file_id, (char *)fixed_string_value, sizeof(int)),
        file_location += PAGE_SIZE;

```



```

        current_position = sizeof(float);
PROCESS_MBIT_TABLE_PART_2( large_neg_integer_rem, BITMAP_INTEGER_SIZE, 1 )
break;
case medium_integer:
PROCESS_MBIT_TABLE_PART_1( medium_integer_rem, unsigned short * ),
5       fseek(file_id, file_location, 0);
       sprintf(fixed_string_value, "%4d", (short *)medium_integer_rem);
       write(file_id, (char *)fixed_string_value, sizeof(int));
       file_location += PAGE_SIZE;
       current_position = sizeof(int);
PROCESS_MBIT_TABLE_PART_2( medium_integer_rem, BITMAP_INTEGER_SIZE, 1 );
break;
case fixed_string:
PROCESS_MBIT_TABLE_PART_1( medium_integer_rem, unsigned short * );
10       fseek(file_id, file_location, 0);
       string_value = malloc(field_length+1);
       CHECK_ALLOCATION(string_value "string_value Routine write_value_distribution()");
       strncpy(string_value, fixed_field[field_offset] -> fixed_string[medium_integer_rem],
->string, field_length);
       string_value[field_length] = '\0';
       write(file_id, string_value, field_length);

       free ( string_value );
       string_value = NULL;
       file_location += PAGE_SIZE;
15       current_position = field_length;
PROCESS_MBIT_TABLE_PART_2( medium_integer_rem, BITMAP_INTEGER_SIZE, 1 );
break;
case year_month:
case year_month_day:
PROCESS_MBIT_TABLE_PART_1( medium_integer_rem, unsigned short * ),
       fseek(file_id, file_location, 0);
       sprintf(fixed_string_value, "%s", number_to_date("medium_integer_rem"));
       write(file_id, (char *)fixed_string_value, 10);
20       file_location += PAGE_SIZE;
       current_position = 10;
PROCESS_MBIT_TABLE_PART_2( medium_integer_rem, BITMAP_INTEGER_SIZE, 1 );
break;
case string_type:
PROCESS_MBIT_TABLE_PART_1( string_ptr, unsigned char * ),
       fseek(file_id, file_location, 0);
       string_value = malloc(field_length+1);
       CHECK_ALLOCATION(string_value "string_value Routine write_value_distribution()");
       strncpy(string_value, (char *)string_ptr, field_length);
25       string_value[field_length] = '\0';
       write(file_id, string_value, field_length);

       free ( string_value );
       string_value = NULL;
       file_location += PAGE_SIZE;
       current_position = field_length;
PROCESS_MBIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE * field_length, field_length );
break;
30
}

return(current_position);
}
35

```

```

/*
  For every bit set in the customer bitmap (master_bitmap), the appropriate field value is written to a file
*/
int write_pur_prd_value_dist(file_id, file_location, data, field_type, field_length, field_offset, value_bitmap,
                             query_mode, query_type, pur_prd_count)
5  int file_id;
  int file_location;
  struct dataset *data;
  enum data_type field_type;
  unsigned int field_length;
  int field_offset;
  struct bitmap *value_bitmap;
  enum pur_prd_type query_mode;
  enum pur_prd_query_type query_type;
  int *pur_prd_count;
10 {
    unsigned int i, j, k, l, m=0, mm;
    unsigned int num_bits, num_values=1;
    unsigned long *long_item;
    int *large_neg_integer_item, neg_integer_value;
    unsigned int *large_integer_item, integer_value;
    unsigned short *medium_integer_item, medium_integer_value;
    unsigned char *small_integer_item;
    unsigned int *bit_item;
15  unsigned int temp_counter;
    unsigned short temp_map;
    float *floating_item, float_value;
    double *double_item, double_value;
    char *string_value;
    char *string_ptr;
    char fixed_string_value[FIXED_STRING_LENGTH];
    int current_position=0;

20  struct value_node *root_node = NULL;
    struct value_node *tail_node = NULL;

    string_value = malloc(field_length+1);
    CHECK_ALLOCATION(string_value, "string_value:show_value_distribution()");

    temp_counter = value_bitmap->start;

    if (query_mode == purchase)
        temp_map = pur01_map;
25  else
        if (query_mode == product)
            temp_map = prd01_map;

    num_bits = value_bitmap->number_of_bits;

    switch (field_type)
    {
        case character
30  PROCESS_PBIT_TABLE_PART_1( small_integer_item, unsigned char * );
        (*pur_prd_count)++;
        lseek(file_id, file_location, 0);
        sprintf(fixed_string_value, "%c", (*(small_integer_item+mm)));
        write(file_id, fixed_string_value, sizeof(int));
        file_location += PAGE_SIZE;
        current_position = sizeof(int);
        PROCESS_PBIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;

        case small_integer
35  PROCESS_PBIT_TABLE_PART_1( small_integer_item, unsigned char * );
        (*pur_prd_count)++;

```

```

        lseek(file_id.file_location,0);
        sprintf(fixed_string_value,"%4d", (*(small_integer_item+mm)));
        write(file_id, (char *)fixed_string_value, sizeof(int));
        file_location += PAGE_SIZE;
        current_position = sizeof(int);
5      PROCESS_PBIT_TABLE_PART_2( small_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case bit_type
    case large_integer
        PROCESS_PBIT_TABLE_PART_1( large_integer_item, unsigned int * );
        (*pur_pro_count)++;
        lseek(file_id.file_location,0);
        sprintf(fixed_string_value,"%10d", (*(large_integer_item+mm)));
        write(file_id, (char *)fixed_string_value, field_length);
        file_location += PAGE_SIZE;
10      current_position = sizeof(int);
        PROCESS_PBIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case dollars
        PROCESS_PBIT_TABLE_PART_1( large_integer_item, unsigned int * );
        (*pur_pro_count)++;
        lseek(file_id.file_location,0);
        integer_value = (*(large_integer_item+mm));
        double_value = (double)integer_value/100.0;
        sprintf(fixed_string_value,"%7.2f", (double)double_value);
15      write(file_id, (char *)fixed_string_value, sizeof(int));
        file_location += PAGE_SIZE;
        current_position = sizeof(double);
        PROCESS_PBIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case floating_point
        PROCESS_PBIT_TABLE_PART_1( large_integer_item, unsigned int * );
        (*pur_pro_count)++;
        lseek(file_id.file_location,0);
        integer_value = (*(large_integer_item+mm));
        float_value = (float)integer_value/100.0;
20      sprintf(fixed_string_value,"%7.2f", (float)float_value);
        write(file_id, (char *)fixed_string_value, sizeof(int));
        file_location += PAGE_SIZE;
        current_position = sizeof(float);
        PROCESS_PBIT_TABLE_PART_2( large_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
25      case negative_float
        PROCESS_PBIT_TABLE_PART_1( large_neg_integer_item, int * );
        (*pur_pro_count)++;
        lseek(file_id.file_location,0);
        neg_integer_value = (*(large_neg_integer_item+mm));
        float_value = (float)neg_integer_value/100.0;
        sprintf(fixed_string_value,"%7.2f", (float)float_value);
        write(file_id, (char *)fixed_string_value, sizeof(int));
        file_location += PAGE_SIZE;
        current_position = sizeof(float);
30      PROCESS_PBIT_TABLE_PART_2( large_neg_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case medium_integer
        PROCESS_PBIT_TABLE_PART_1( medium_integer_item, unsigned short * );
        (*pur_pro_count)++;
        lseek(file_id.file_location,0);
        sprintf(fixed_string_value,"%4d", (*(medium_integer_item+mm)));
        write(file_id, (char *)fixed_string_value, sizeof(int));
        file_location += PAGE_SIZE;
        current_position = sizeof(int);
35      PROCESS_PBIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 );
        break;
    case fixed_string

```

```

PROCESS_PBIT_TABLE_PART_1( medium_integer_item, unsigned short * ).
(*pur_pro_count)++;
    lseek(file_id, file_location, 0);
    string_value = malloc(field_length+1);
    CHECK_ALLOCATION(string_value, "string_value Routine write_value_distribution()");
    strcpy(string_value, fixed_field[field_offset]->fixed_string[("medium_integer_item+mm")]);
    string_value[field_length]='\0';
    write(file_id, string_value, field_length);

    free ( string_value );
    string_value = NULL;
    file_location += PAGE_SIZE;
    current_position = field_length;
PROCESS_PBIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 ).
break;
case year_month:
case year_month_day
PROCESS_PBIT_TABLE_PART_1( medium_integer_item, unsigned short * ).
(*pur_pro_count)++;
    lseek(file_id, file_location, 0);
    sprintf(fixed_string_value, "%s", number_to_date(("medium_integer_item+mm")));
    write(file_id, (char *)fixed_string_value, 10);
    file_location += PAGE_SIZE;
    current_position = 10;
PROCESS_PBIT_TABLE_PART_2( medium_integer_item, BITMAP_INTEGER_SIZE, 1 ).
break;
case string_type:
PROCESS_PBIT_TABLE_PART_1( string_ptr, unsigned char * ).
(*pur_pro_count)++;
    lseek(file_id, file_location, 0);
    string_value = malloc(field_length+1);
    CHECK_ALLOCATION(string_value, "string_value Routine write_value_distribution()");
    strcpy(string_value, string_ptr+(mm*field_length), field_length);
    string_value[field_length]='\0';
    write(file_id, string_value, field_length);

    free ( string_value );
    string_value = NULL;
    file_location += PAGE_SIZE;
    current_position = field_length;
PROCESS_PBIT_TABLE_PART_2( string_ptr, BITMAP_INTEGER_SIZE*field_length, field_length ).
break;
}

return(current_position)
}

THIS ROUTINES HAVE BEEN OUT OF SYNC WITH THE DISTRIBUTION, PLEASE CHECK THE VALIDITY.
/*
This routine is not being used currently
This routine writes multiple field values on to a file. However, the values are written one row at a time.
/*
int write_field_distribution()
{
    int status;
    int chan;

```

```

    struct address_range reladdr;
    struct query_info *tempq;
    struct field_entry *field;
    int file_id;
    int file_location, old_location=0, offset=15;
    int i, num_records;
5   char null_buffer[PAGE_SIZE];
    struct value_node *distribution;
    char *buffer;
    char msg_buffer[132];
    int field_offset=0;
    struct query_info *query_tree_head= NULL;
    int length;
    char length_str[80];
    int reference_count=0;
10  enum pur_pro_query_type query_type;
    enum pur_pro_type query_mode;
    unsigned int pur_pro_count=0;

    length = get_field_length();
    sprintf(length_str, "%s%d", "mrs=", length);

    if (master_count == 0)
15  {
        strcpy(msg_buffer, "Query did not produce any results.\n");
        strcat(msg_buffer, "No distribution created.\n\n");
        if (DMO_CONNECTED)
            msg_buf_xfer(msg_buffer);
        else
            printf("%s", msg_buffer);

        return(SUCCESS);
    }
20  else
        num_records = count_set_bits(master_bimap);

    for(i=0; i<PAGE_SIZE-1; i++)
        null_buffer[i] = '\0';
    null_buffer[i] = '\n';

    file_id = open("write_fields" FILE_READ_WRITE, READ_WRITE_MODE);
25  if (file_id < 0)
    {
        if (errno == ENOENT)
        {
            file_id = creat("select_fields", FILE_WRITE, length_str);
            if (file_id < 0)
            {
                error_handler(FILE_CREATE_ERR, ERROR, NO_STATUS, "select_fields");
            }
            else
30  {
                for(i=0; i<num_records; i++)
                    write(file_id, null_buffer, PAGE_SIZE);
            }
        }
    }

    /* the filename is built in this manner */
    sprintf(msg_file_buffer, "write_dir %s", query_filename);
35  file_id = creat(msg_file_buffer, FILE_WRITE, length_str);
    if (file_id < 0)
    {

```

```

    if (!DMO_CONNECTED)
        error_handler(FILE_CREATE_ERR, ERROR, NO_STATUS, "write_fields");
    else
    {
        /* to account for the two header lines */
        /* the first line is the field_name and the second line is the underline */
        num_records += 2;
        for(i=0; i<num_records; i++)
            write(file_id, null_buffer, PAGE_SIZE);
    }

    while((tempq = next_parse_entry()) != NULL)
    {
        if (query_tree_head == NULL)
            query_tree_head = tempq;
        else
        {
            free_parse_tree(query_tree_head);
            query_tree_head = tempq;
        }
    }

    field = tempq->field;
    /* map section file */
    status = OpenMapFile(field->table, "tbodyar.sec", field->field_name,
                        &chan, &retadr, field->von, field->number_of_blocks);
    if (is_error(status))
        error_handler(MAP_OPEN_ERR, ERROR, status, "write_field_distribution");

    /* set start address of data points */
    field->data->nems = retadr.start;

    if (field->field_type == fixed_string)
        field_offset = get_fixed_string_offset(field->field_name);

    file_location = old_location;
    /* search for value distribution */
    /* update this routine to do the writing as well */
    /* you need to pass the file_id and file_location */
    /* take a look at write_value_tree (ust.c) to see how the write is done and put the same code in this routine */
    lseek(file_id, file_location, 0);
    write(file_id, (char *)field->field_label, strlen(field->field_label));
    file_location += PAGE_SIZE;

    lseek(file_id, file_location, 0);
    write(file_id, (char *)UNDERLINE, strlen(field->field_label));
    file_location += PAGE_SIZE;

    if (((strcmp(field->field_name, "CUS", 3)) != 0) &&
        ((purchase_query && (strcmp(field->field_name, "PUR", 3)) == 0)) &&
        ((product_query && (strcmp(field->field_name, "PRD", 3)) == 0))))
    {
        sscanf(field->field_name+3, "%d", &reference_count);

        if ((strcmp(field->field_name, "SUB", 3)) == 0)
        {
            file_location = write_subs_value_distribution(file_id, file_location, field->data,
                field->field_type, field->field_end, field_offset, master_bitmap,
                subsidiary_bitmap[reference_count]);
        }
        else
    }
}

```

```

    if ((strcmp(field->field_name, "PUR", 3)) == 0)
        query_mode = purchase;
    else
        if ((strcmp(field->field_name, "PRD", 3)) == 0)
5         query_mode = product;

    query_type = reference_count;

    switch (query_type)
    {
        case avg_pur_prd:
        case total_pur_prd:
            strcpy(msg_buffer, "\n\nNOTE:\n");
            if (query_mode == purchase)
10             strcat(msg_buffer, "Distribution for Average or Total Purchase has not yet been implemented\n");
            else
                strcat(msg_buffer, "Distribution for Average or Total Product has not yet been implemented\n");
            strcat(msg_buffer, "Please contact the Database Link Product Manager for release dates\n\n");
            if (DMQ_CONNECTED)
                msg_buf_xfer(msg_buffer);
            else
                printf("%s", msg_buffer);
            return(SUCCESS);
15         }
        file_location = write_pur_prd_value_dist(file_id, file_location, field->data,
            field->field_type, field->field_end, field_offset, master_bitmap, query_mode, query_type,
            &pur_prd_count);
    }
    else
    {
        file_location = write_value_distribution(file_id, file_location, field->data,
20         field->field_type, field->field_end, field_offset, master_bitmap);
    }

    file_location += offset;
    old_location += file_location;

    /* unmap section file */
    status = UnmapCloseFile(chan, &retaddr);
    if (is_error(status))
        error_handler (UNMAP_CLOSE_ERR ERROR status "write_field_distribution")
25 }
    close(file_id);

    if (query_tree_head != NULL)
    {
        free_parse_tree(query_tree_head);
        free(query_tree_head);
    }

30 return(SUCCESS);
}

/*
   This routine searches the segmentation field values in the database and for every non-zero value in the
   database a bit is set.
*/
struct bitmap *generate_data_bitmap(value, field_name, file_offset, value_bitmap)
unsigned short      value;
char                field_name;
35 unsigned int      file_offset;
struct bitmap      *value_bitmap;
{

```

```

    unsigned int      i, j, k;
    unsigned int      jj, kk;
    unsigned int      num_bits;
    unsigned short    *medium_integer_ptr;
    unsigned int      *temp_counter;
5   int              max_count;
    struct bitmap     *temp_bitmap;

    if ((temp_bitmap = create_bitmap(value_bitmap->number_of_bits)) == NULL)
        error_handler (BITMAP_NOT_CREATE, ERROR, NO_STATUS, "Value in <generate_data_bitmap>");

    temp_counter = temp_bitmap->start;
    num_bits = temp_bitmap->number_of_bits;

10   kk = BITMAP_INTEGER_SIZE;
    jj = 1;

    for (i=0; i<num_bits; i++)
    {
        if (*value)
        {
            *temp_counter |= jj;
        }
        value++;

15         if (--kk)
            jj <<= 1;
        else
        {
            temp_counter++;
            jj = 1;
            kk = BITMAP_INTEGER_SIZE;
        }
    }

20   return(temp_bitmap);
}

/*
   This routine resets the database values for this field to zero
*/
reset_segmentation_value(value, field_name, file_offset, value_bitmap)
unsigned short    *value;
char              *field_name;
25   unsigned int    file_offset;
    struct bitmap   *value_bitmap;
{
    unsigned int    i, j, k;
    unsigned int    jj, kk;
    unsigned int    num_bits;
    unsigned short  *medium_integer_ptr;
    unsigned int    *temp_counter;
    int              max_count;

30   temp_counter = value_bitmap->start;
    num_bits = value_bitmap->number_of_bits;

    kk = BITMAP_INTEGER_SIZE;
    jj = 1;

    for (i=0; i<num_bits; i++)
    {
        if (*temp_counter)
        {
35         if (*temp_counter & jj)
            *value = 0;

            value++;
        }
    }
}

```



```

        if ( -kk )
            y <= 1;
        else
5         {
            temp_counter++;
            y = 1;
            kx = BITMAP_INTEGER_SIZE;
        }
    }
    /* temp_counter == 0, bump past whole word */
    else
    {
10         value += BITMAP_INTEGER_SIZE;
            i += BITMAP_INTEGER_SIZE - 1;
            temp_counter++;
    }
}

/*
15  After the segmentation query is executed, the fixed_string array is then written back to the file
*/
print_fixed_string_list(field_name, file_offset, max_count)
char field_name;
unsigned int file_offset;
int max_count;
{
    int i;
    char filename[FILE_NAME_LENGTH];
    FILE *f;

20     sprintf(filename, "FIXED_FILE_DIR %s.idx", field_name);
    if ((f = fopen(filename, "w")) != NULL)
    {
        /* read the fixed string file information */
        fprintf(f, "%d\n", max_count);
        for (i = 1; i <= max_count; i++)
            fprintf(f, "%d\n", i, fixed_field[file_offset] -> fixed_string[i] -> string);
    }
    else
25     {
        error_handler(FILE_NOT_OPEN, ERROR, NO_STATUS, filename);
    }
    if (fclose(f) != EOF)
        error_handler(FILE_NOT_CLOSE, ERROR, NO_STATUS, filename);
}

/*
30  This routine open the fixed string file and returns the (max_count+1) value. Essentially providing the next value that
    can be occupied.
*/
int get_fixed_string_max_count(field_name)
char field_name;
{
    int i, max_count = 0;
    char filename[FILE_NAME_LENGTH];
    FILE *f;

    sprintf(filename, "FIXED_FILE_DIR %s.idx", field_name);
    if ((f = fopen(filename, "r")) != NULL)
35         /* read the fixed string file information */
            fscanf(f, "%d", &max_count);
    else
        error_handler(FILE_NOT_OPEN, ERROR, NO_STATUS, filename);
    if (fclose(f) != EOF)
        error_handler(FILE_NOT_CLOSE, ERROR, NO_STATUS, filename);
    return(++max_count);
}

```

```

/*
  This routine is currently not being used
  Intended to give the next value in the fixed_field[field_offset]->fixed_string array that is not occupied (=NULL)
*/
5 int get_fixed_string_max_count(file_offset)
  unsigned int file_offset;
{
  int i=0;
  int fixed_count=0;

  while (fixed_field[file_offset]->fixed_string[i++] != NULL)
    fixed_count++;

  return(++fixed_count);
10 }

/*
  insert_segmentation_information() - will check the super_master_bitmap, and for every bit set,
  the database will be updated. And, in addition, the fixed
  string file associated with the field will be updated with
  the indx (i.e. the 1 or 2, depending on the value in the DB)
  and the keycode information.

  Note: Initially the database will be updated with a 1 and all
  subsequent hits will increment the count.
15

*/
insert_segmentation_information(value, field_name, file_offset, value_bitmap, query_keycode_upper)
unsigned short      *value;
char                *field_name;
unsigned int        file_offset;
struct bitmap      *value_bitmap;
char                *query_keycode_upper;
20 {
  unsigned int      i, j, k;
  unsigned int      jj, kk;
  unsigned int      num_bits;
  unsigned short    *medium_integer_item;
  unsigned int      temp_counter;
  int               max_count;
  int               bit_set=0;

  if (file_offset == -1)
25     file_offset = create_fixed_string(field_name);

  if (reset_segmentation)
    /* set the index to 1 */
    max_count = 1;
  else
    /* get the last index */
    max_count = get_fixed_string_max_count(field_name);

  temp_counter = value_bitmap->start;
  num_bits = value_bitmap->number_of_bits;

  kk = BITMAP_INTEGER_SIZE;
  jj = 1;

  for (i=0; i<num_bits; i++)
  {
    if (temp_counter)
    {
      if (temp_counter & jj)
      {
        if (*value == 0)
        {
35

```

```

        bit_set = 1;
        *value = max_count;
        if (fixed_field[file_offset] -> fixed_string[*value] == NULL)
        {
            fixed_field[file_offset] -> max_count++;
            fixed_field[file_offset] -> fixed_string[*value] = (struct fixed_string_value_type *)
                malloc (sizeof(struct fixed_string_value_type));

            CHECK_ALLOCATION(fixed_field[file_offset] -> fixed_string[*value]);
            *fixed_field[file_offset] -> fixed_string Routine insert_segmentation_information("");
            strcpy(fixed_field[file_offset] -> fixed_string[*value] -> string, query_keycode_upper);
        }
        else
            strcpy(fixed_field[file_offset] -> fixed_string[*value] -> string, query_keycode_upper);
    }
    else
        error_handler (COLUMN_DATA_ERR, ERROR, NO_STATUS, NO_STRING);

        fixed_field[file_offset] -> fixed_string[*value] -> bit_select = 0;
    }

    value++;

    if (--kk)
        jj <= 1;
    else
    {
        temp_counter++;
        jj = 1;
        kk = BITMAP_INTEGER_SIZE
    }
}
/* Temp_counter == 0. bump past whole word */
else
{
    value += BITMAP_INTEGER_SIZE;
    i += BITMAP_INTEGER_SIZE - 1;
    temp_counter++;
}

}

if (bit_set)
    /* print the values to file */
    print_fixed_string_list(field_name file_offset, max_count);
}

/*
Create_segmentation() - will select the section file of the field selected, and update
the bns.

*/
30 create_segmentation(field)
    struct field_entry *field;
{
    int status;
    int chan;
    struct address_range retaddr;
    unsigned int field_offset = -1;
    struct bitmap *temp_bitmap, *data_bitmap;
    char *query_keycode_upper;
    35 int segment_count;
    char buffer[132];

```

```

if ((field->field_type == fixed_string) &&
    (strcmp(field->field_name, "CUS_SEGX", 8) == 0))
{
    status = WriteMapFile(field->table, "toobar.sec", field->field_name,
        &chan, &readdr, field->vbn, field->number_of_blocks);

5      if (is_error(status))
        error_handler (WRITE_MAP_ERR, ERROR, status, "create_segmentation");

        /* set start address of data points */
        field->data->items = readdr.start;

        field_offset = get_fixed_string_offset(field->field_name);

        query_keycode_upper = strtoupper(query_keycode);

10      if (reset_segmentation)
        {
            if ((temp_bitmap = create_bitmap(super_master_bitmap->number_of_bits)) == NULL)
                error_handler (BITMAP_NOT_CREATE, ERROR, NO_STATUS, "Super_master in <create_segmentation>");
            printf("Initialized bitmap count is %d\n", count_set_bits(temp_bitmap));

            /* the complement of the bitmap is done, so that i can initialize the data field */
            temp_bitmap = complement_bitmap(temp_bitmap);
            printf("Complemented bitmap count is %d\n", count_set_bits(temp_bitmap));

15      reset_segmentation_value(field->data->items, field->field_name, field_offset, temp_bitmap);
            printf("Reset bitmap count is %d\n", count_set_bits(temp_bitmap));

            /* insert segmentation information */
            insert_segmentation_information(field->data->items, field->field_name, field_offset,
                super_master_bitmap,
                                query_keycode_upper);

            /* svp - changes need to be verified */

20      /*
            sprintf (buffer, "\n\n Segmentation Bits set for %s is %d\n", query_keycode_upper, master_count);
            if (DMO_CONNECTED)
                msg_buf_xfer (buffer);
            else
                printf("%s", buffer);
        */
        }
    else
    {
25      /* create a bitmap of the data_column */
        data_bitmap = generate_data_bitmap(field->data->items, field->field_name, field_offset, master_bitmap);
        printf("Generate data bitmap count is %d\n", count_set_bits(data_bitmap));

        temp_bitmap = complement_bitmap(data_bitmap);
        printf("Complement bitmap count is %d\n", count_set_bits(temp_bitmap));

        if (combine_bitmaps(temp_bitmap and master_bitmap) == FAILURE)
            error_handler (BITMAP_NOT_COMBINE, ERROR, NO_STATUS,
                "Master AND Temp in <create_segmentation>");
        segment_count = count_set_bits(temp_bitmap);
        printf("And with the master_bitmap, the bitmap count is %d\n", segment_count);

30      /* svp - changes need to be verified */

        sprintf (buffer, "\n\n Segmentation Bits set for %s is %d\n", query_keycode_upper, segment_count);
        if (DMO_CONNECTED)
            msg_buf_xfer (buffer);
        else
35      printf("%s", buffer);
    }
}

```

```

/* insert segmentation information */
insert_segmentation_information(field->data->items, field->field_name, field_offset, temp_bimap,
5 query_keycode_upper);

}

/* unmap section file */
status = UnmapCloseFile(chan, &retaddr);
if (is_error(status))
    error_handler (UNMAP_CLOSE_ERR, ERROR, status, "create_segmenation");

10 free(query_keycode_upper);
   query_keycode_upper = NULL;

   return(SUCCESS);
}
else
{
    error_handler (SEGMENTATION_ERR, ERROR, NO_STATUS, NO_STRING);

15

20

25

30

35
```

5

Appendix D for U.S. Patent Application

DATABASE LINK SYSTEM
BACKGROUND OF THE INVENTION

Filed October 22, 1993

10

Appendix D: Set of Routines which Manipulate Bitmaps which
Used in Holding Results of Queries Against Databases.

© 1993 FDC, Inc.
All Rights Reserved

15

20

25

30

35

```

extern int cus_pur_chnl;      /* OPENMAPFILE channel for customer to purchase map count data */
extern int cus_pro_chnl;      /* OPENMAPFILE channel for customer to product map count data */
extern int pur_pro_chnl;      /* OPENMAPFILE channel for purchase to product map count data */

5  extern struct address_range cus_pur_addr; /* addresses of mapped customer to purchase map count data for OPENMAPFILE.
   UNMAPCLOSE */
   extern struct address_range cus_pro_addr; /* addresses of mapped customer to product map count data for OPENMAPFILE.
   UNMAPCLOSE */
   extern struct address_range pur_pro_addr; /* addresses of mapped purchase to product map count data for OPENMAPFILE.
   UNMAPCLOSE */

/* flush flags. mark if map count data is mapped or not. Always 0, unless corresponding delete_pur_info flag is on, for
allowing dynamic map and unmap. (if delete_pur_info is 0, then map always mapped and these flag stay at 0 */
extern int cus_pur_map;      /* flush flags. 1=mapped. 0=not mapped. customer to purchase map count data */
extern int cus_pro_map;      /* flush flags. 1=mapped. 0=not mapped. customer to product map count data */
extern int pur_pro_map;      /* flush flags. 1=mapped. 0=not mapped. purchase to product map count data */

10

/* Module contains the following routines:
struct bitmap *create_bitmap(number_of_items);
void free_bitmap(bitmap);
int count_set_bits(bitmap);
int validate_bitmaps_for_operation(map_one, map_two);
int combine_bitmaps(master_map, conjunction, map_two);
void *copy_bitmaps(map_one, map_two);
int getbit(word, n);
15 int print_bits(word, size);
   int write_bits(filename, word, size);
   int display_bitmap(bitmap);
   int write_bitmap(bitmap, filename);
   void load_bitmaps();
   void load_pur_pro_maps();

*/

20 /* Bitmap structure
   --
   -- number_of_bits = number of bits used in map
   -- number_of_integers = number of unsigned integers used to make the bitmap
   --
   --
   -- start points to      int 1
   --                      int 2
25 --                      int 3
   --
   --
   -- end points to      last int
   --
   -- end_bits = number of bits used for bitmap in integer pointed to by end;
   --
   -- number_of_bits = (number_of_integers * BITMAP_INTEGER_SIZE) - (BITMAP_INTEGER_SIZE - end_bits)
30 --
   --
   --
   -- CREATE_BITMAP
   --
   -- create bitmap will allocate and initialize the necessary space for a
   -- bitmap structure based on the customer record size
35 -- FORMAT
   --

```

```

/*
--
-- MODULE: BITMAPS.C
--
-- MODULE DESCRIPTION:
--
-- Set of routines which manipulate bitmaps which are used in
5 -- holding results of queries against datasets.
--
-- AUTHORS:
--
-- Kelly Westman    Digital Equipment Corporation
-- Sushil Pillai    Digital Equipment Corporation
--
-- CREATION DATE: 7-August-1992
--
-- DESIGN ISSUES:
10 --
-- Could add several return codes for some of the routines rather than
-- just SUCCESS or FAILURE
--
-- PORTABILITY ISSUES:
--
-- None
--
-- MODIFICATION HISTORY:
15 --
-- 7-August-1992 - Original
-- 8-Sept-1993 - Chuck Malmeskog
--      - Free_bitmap cleanup
--      - Removed #ifdef DEBUG to make code readable again
--      - Added comments to clarify how bitmap is put together
-- 30-Sep-1993 - Charles Malmeskog
--      - Update error_handler() calls to include ERROR message type
--
*/

20
/*
--
-- INCLUDE FILES
--
*/
#include <stdio>
#include "tcc_prototype"
#include "tcc_multi_file"
#include "tcc_error_numbers.h" /* for error_handler information */

25
/* External Declarations */
extern int max_number_of_bits
extern int max_purchase_bits
extern int max_product_bits

extern int bit_counts[256].

/* for purchase and product selection */
30
extern int sel_subsidary;
extern int sel_purchase;
extern int sel_product;

extern unsigned int delete_pur_info;
extern unsigned int delete_pro_info;
extern unsigned int delete_pur_pro_info;

35

```



```

-- struct bitmap *create_bitmap(unsigned int number_of_items)
--
-- ARGUMENTS
--
5      -- number_of_items
--
--      the number of items which will be held in the bitmap. Each item
--      is given a bit. Therefore number of items = number of active bits in
--      the bitmap.
--
-- RETURN VALUES
--
--      x      - a pointer to the bitmap structure created.
--      NULL   - the bitmap was not created.
10     */

struct bitmap *create_bitmap(number_of_items)
unsigned int number_of_items;
{
    unsigned int *start = NULL;
    unsigned int *end = NULL;
    unsigned short end_bits = 0;
    unsigned int number_to_allocate = 0;
    unsigned int number_of_bits;
15
    struct bitmap *temp;

    number_of_items = max_number_of_bits;

    /* make sure this is a meaningful call */
    if( number_of_items == 0 )
        return(NULL);
20
    number_of_bits = number_of_items;

    /* determine number to allocate for bitmap */
    number_to_allocate = number_of_bits/BITMAP_INTEGER_SIZE;

    /* add additional integer for remaining bits and set end_bits */
    if(end_bits = (number_of_bits % BITMAP_INTEGER_SIZE))
25     {
        number_to_allocate++;
    }
    else
    {
        end_bits = BITMAP_INTEGER_SIZE;
    }

    start = (unsigned int *) calloc(number_to_allocate * sizeof(unsigned int));
    CHECK_ALLOCATION(start, "start, Routine create_bitmap()");
30
    end = start + (number_to_allocate - 1);

    /* allocate structure for return */
    temp = (struct bitmap *) malloc(sizeof(struct bitmap));
    CHECK_ALLOCATION(temp, "temp, Routine create_bitmap()");

    temp->start = start;
    temp->end = end;
    temp->number_of_bits = number_of_bits;
    temp->end_bits = end_bits;
35
    return(temp);
}

```

```

/*
 * CREATE_GENERAL_BITMAP
 *
 * create general bitmap will allocate and initialize the necessary space for a
 * bitmap structure.
5  *
 * FORMAT
 *
 * struct bitmap *create_general_bitmap(unsigned int number_of_items)
 *
 * ARGUMENTS
 *
 * number_of_items
 *
 * the number of items which will be held in the bitmap. Each item
10 * is given a bit. Therefore number of items = number of active bits in
 * the bitmap.
 *
 * RETURN VALUES
 *
 * x - a pointer to the bitmap structure created.
 * NULL - the bitmap was not created.
 */

15 struct bitmap *create_general_bitmap(number_of_items)
   unsigned int number_of_items;
{
   unsigned int *start = NULL;
   unsigned int *end = NULL;
   unsigned short end_bits = 0;
   unsigned int number_to_allocate = 0;
   unsigned int number_of_bits;

20   struct bitmap *temp;

   /* make sure this is a meaningful call */
   if( number_of_items == 0 )
      return(NULL);

   number_of_bits = number_of_items;

   /* determine number to allocate for bitmap */
25   number_to_allocate = number_of_bits/BITMAP_INTEGER_SIZE;

   /* add additional integer for remaining bits and set end_bits */
   if(end_bits = (number_of_bits % BITMAP_INTEGER_SIZE))
   {
      number_to_allocate++;
   }
   else
   {
30     end_bits = BITMAP_INTEGER_SIZE;
   }

   start = (unsigned int *) calloc(number_to_allocate, sizeof(unsigned int));
   CHECK_ALLOCATION(start, "start, Routine create_general_bitmap()");

   end = start + (number_to_allocate - 1);

   /* allocate structure for return */
   temp = (struct bitmap *) malloc(sizeof(struct bitmap));
35   CHECK_ALLOCATION(temp, "temp, Routine create_general_bitmap()");

   temp->start = start;
   temp->end = end;
   temp->number_of_bits = number_of_bits;
   temp->end_bits = end_bits;

   return(temp);
}

```

```

/
-- FREE_BITMAP
--
-- Free_bitmap will de-allocate the memory for a bitmap structure, and
-- set the bitmap structure pointer to NULL.
5 -- FORMAT
--
-- struct bitmap *create_bitmap(unsigned int number_of_items)
--
-- ARGUMENTS
--
-- structure bitmap *map
--     A pointer to a bitmap structure which will be de-allocated.
10 -- RETURN VALUES
--
-- NONE
--
*/

void free_bitmap(map)
struct bitmap *map;
{
15     if(map == NULL)
    {
        error_handler(FREE_NULL_PTR, ERROR, map, "free_bitmap in bitmaps.c");
    }
    else
    {
        if(map->start != NULL)
        {
20             cfree(map->start);
            map->start = NULL;
            map->end = NULL;
        }

        free(map);
        map = NULL;
    }
    return;
}

25 /
-- COMPLEMENT_BITMAP
--
-- complement_bitmap will produce a complement of the specified bitmap
--
-- FORMAT
--
-- struct bitmap *complement_bitmap(struct bitmap *map)
--
30 -- ARGUMENTS
--
-- structure bitmap *map
--     a pointer to a bitmap structure which will be de-allocated
--
-- RETURN VALUES
--
-- NONE
--
*/

35 struct bitmap *complement_bitmap(map)
struct bitmap *map;
{
    ?

```

```

int j;
int *temp;
struct bitmap *comp_map;

comp_map = map;

5   temp = map->start;
    while (temp < map->end)
    {
        *temp = ~(*temp);
        temp++;
    }
    temp = map->end;

10  for (j=0; j<map->end_bits; j++)
    {
        if (*temp & setbit(j))
            *temp &= ~(setbit(j));
        else
            *temp |= setbit(j);
    }

    return(comp_map);
}

15  /
    COUNT_SET_BITS
    /
    count_set_bits will determine the total number of bits set to 1
    in a bitmap.
    /
    FORMAT
    /
20  int count_set_bits(struct bitmap *map)
    /
    ARGUMENTS
    /
    struct bitmap *map,
    a pointer to a bitmap structure for which the total number of
    set bits will be counted
    /
    RETURN VALUES
    /
25  n      - the total number of set bits
    FAILURE - the operation is not valid on this bitmap.
    /

    /
    New algorithm of Mike Emerson's. counts 8 bits at a time
    /
    Note: this algorithm works for 32 bit int bitmaps, and
    uses knowledge that there are 4 8 bit bytes in each int
30  With different size variables, this would need revisions
    /

int count_set_bits(map)
struct bitmap *map;
{
    unsigned int j, jj, kk, count;

35  union
    {

```

```

    int aa;
    unsigned char bb[4];
} val;

    int total_count = 0;

5   unsigned int temp;

    if(map == NULL)
        return(FAILURE);

    /* where start < end, there are always BITMAP_INTEGER_SIZE bits to count */
    temp = map->start;

10  while(temp < map->end)
    {
        if( val.aa == temp )
        {
            total_count += bit_counts[val.bb[0]];
            total_count += bit_counts[val.bb[1]];
            total_count += bit_counts[val.bb[2]];
            total_count += bit_counts[val.bb[3]];
        }
        temp++;
15  }

    /* last word in table may have less than 32 valid bits, be sure
    all bits beyond end_bits are off */
    if( val.aa == map->end )
    {
        if( (count=map->end_bits) < BITMAP_INTEGER_SIZE )
        {
            jj = 1;
            kk = 0;
20      for( j=0; j<count; j++)
            {
                kk |= jj;
                jj <<= 1;
            }
            /* kk will have all bits from 0 to count set */
            /* following and operation will guarantee all bits beyond end_bits are 0 */
            val.aa &= kk;
25  }

    /* same operation with last word in table */
    total_count += bit_counts[val.bb[0]];
    total_count += bit_counts[val.bb[1]];
    total_count += bit_counts[val.bb[2]];
    total_count += bit_counts[val.bb[3]];
    }
    return(total_count);

30  }

/*
-- VALIDATE_BITMAPS_FOR_OPERATION
-- VALIDATE_BITMAPS_FOR_OPERATION will check two bitmaps for compatibility
-- for operations on them. It will make sure they are of the same size and
-- not NULL
--
-- FORMAT
--
35  -- int validate_bitmaps_for_operations(struct bitmap *map_one,
--                                     struct bitmap *map_two)
--
-- ARGUMENTS
--

```

```

-- struct bitmap *map_one;
--     a pointer to a bitmap structure
--
-- struct bitmap *map_two;
--     a pointer to a bitmap structure.
5  --
-- RETURN VALUES
--
-- SUCCESS - bitmap map_one and bitmap map_two can be
--          combined or copied because they are the same size/ not null
-- FAILURE - do not use the two maps together
*/

10 int validate_bitmaps_for_operation(map_one, map_two)
    struct bitmap *map_one;
    struct bitmap *map_two;
    {

        /* make sure no one is doing something they shouldn't */

        if ( map_one == NULL || map_two == NULL )
        {
15             printf("<validate_bitmaps> Error - Invalid pointer.\n");
            return(FAILURE);
        }

        if ( map_one->number_of_bits != map_two->number_of_bits )
        {
            printf("<validate_bitmaps> Error - Bitmaps differ in size.\n");
            return(FAILURE);
        }

20     return(SUCCESS);
    }

--
-- COMBINE_BITMAPS
--
-- Combine_bitmaps will take two bitmaps and perform a boolean operation
-- on the two, putting the results into the first bitmap
--
-- FORMAT
25 --
-- int combine_bitmaps(struct bitmap *master_map,
--                     enum boolean_operator conjunction,
--                     struct bitmap *map_two)
--
-- ARGUMENTS
--
-- struct bitmap *master_map,
--     a pointer to a bitmap structure which has previously been created
30 --     using create_bitmap(). This map will hold the results of the
--     the boolean operation
--
-- enum boolean_operator conjunction,
--     the boolean operator to use. Supported operators are 'and' and 'or'
--
-- struct bitmap *map_two;
--     a pointer to a bitmap structure which has previously been created
--     using create_bitmap(). This map will be left unchanged.
35 --
-- RETURN VALUES
--

```

```

- SUCCESS - bitmaps were successfully operated on
-
- FAILURE - error occurred (should define more here)
*/

5  int combine_bitmaps (master_map, conjunction, map_two)
    struct bitmap *master_map;
    enum boolean_operator conjunction;
    struct bitmap *map_two;
    {
        unsigned int temp;
        unsigned int temp_two;

10      if (validate_bitmaps_for_operation(master_map, map_two) == FAILURE)
            return (FAILURE);

        temp = master_map->start;
        temp_two = map_two->start;

        switch (conjunction)
        {
            case and:
            case and_when:
15              while (temp <= master_map->end)
                {
                    temp = (temp & temp_two);
                    temp++;
                    temp_two++;
                }
                break;

            case or:
            case or_when:
20              while (temp <= master_map->end)
                {
                    temp = (temp | temp_two);
                    temp++;
                    temp_two++;
                }
                break;

            case except:
25              map_two = complement_bitmap(map_two);
              temp_two = map_two->start;
              while (temp <= master_map->end)
                {
                    temp = (temp & temp_two);
                    temp++;
                    temp_two++;
                }
                break;

            default:
30              return (FAILURE);
                break;
        }

        return (SUCCESS);
    }

35

```

```

/*
-- COPY_BITMAPS
--
-- Copy_bitmaps will take a bitmap and copy it to another bitmap
--
-- FORMAT
5  --
--  int combine_bitmaps(struct bitmap *map_one,
--                      struct bitmap *map_two)
--
-- ARGUMENTS
--
-- struct bitmap *map_one:
--   a pointer to a bitmap structure which will be copied into.
--
-- struct bitmap *map_two:
10 --   a pointer to a bitmap structure which will be copied to map_one.
--
-- RETURN VALUES
--
-- SUCCESS - bitmap map_two was successfully copied to map_one.
-- FAILURE - unsuccessful copy
*/

15 void *copy_bitmaps(map_one, map_two)
    struct bitmap *map_one;
    struct bitmap *map_two;
    {
        unsigned int number_to_copy = 0;

        if(validate_bitmaps_for_operation(map_one, map_two) == FAILURE)
20         {
            return(NULL);
        }

        number_to_copy = map_two->number_of_bits/BITMAP_INTEGER_SIZE;

        /* add additional integer for remaining bits and set end_bits */
        if(map_two->number_of_bits % BITMAP_INTEGER_SIZE)
25         {
            number_to_copy++;
        }
        return(memcpy(map_one->start,
                      map_two->start,
                      number_to_copy*sizeof(unsigned int) ));
    }

/*
-- GET_BIT
30 --
-- get_bit will return 0 or 1 to indicate if a specific bit is set in a word
--
-- FORMAT
--
-- int get_bit(unsigned int word, int n).
--
-- ARGUMENTS
--
-- unsigned int word
35 --   the word to check for a specific bit.
--

```



```

5  -- int n
   -- the bit number to check.
   --
   -- RETURN VALUES
   --
   -- 1 - the bit is set
   -- 0 - the bit is not set
   */

   int getbit(word, n)
   unsigned int word;
   int n;
   {
       return((word >> n) & 01);
10  }

   -- PRINT_BITS
   --
   -- print_bits will print the bit pattern of a word of the specified size on
   -- stdout.
   --
   -- FORMAT
   --
15  -- int print_bits(unsigned int word, int size);
   -- ARGUMENTS
   --
   -- unsigned int word
   -- the word to print a bit pattern of.
   --
   -- int size
   -- the size of the word.
   --
20  -- RETURN VALUES
   --
   -- SUCCESS - the printing was successful.
   -- FAILURE - the printing was not successful.
   */

   int print_bits(word, size)
   unsigned int word;
   int size;
25  {
       int i;

       printf(" ");
       for (i = (size - 1); i >= 0; i--)
       {
           if (getbit(word, i) == 0) printf("0");
           else if (getbit(word, i) == 1) printf("1");
           else
30         {
               printf(" error printing bits ");
               return(FAILURE);
           }
       }
       printf(" ");
       return(SUCCESS);
   }
35

```

```

5  /
   -- WRITE_BITS
   --
   -- write_bits will write the bit pattern of a word of the specified size to
   -- a file.
   --
   -- FORMAT
   --
   -- int write_bits(unsigned int word, int size);
   --
   -- ARGUMENTS
   --
   -- unsigned int word
10  -- the word to print a bit pattern of.
   --
   -- int size
   -- the size of the word.
   --
   -- RETURN VALUES
   --
   -- SUCCESS - the printing was successful.
   -- FAILURE - the printing was not successful.
15  */

int write_bits(filename, word, size)
char *filename;
unsigned int word;
int size;
{
20  int i;
   char *bitfile;
   unsigned int temp;
   FILE *f;

   temp = (struct bitmap *) malloc(sizeof(struct bitmap));
   CHECK_ALLOCATION(temp, "temp, Routine write_bits()");

   bitfile = (char *) malloc(FILE_NAME_LENGTH + 1);
   CHECK_ALLOCATION(bitfile, "bitfile, Routine write_bits()");

   strcpy(bitfile, filename);
25  f = fopen(bitfile, "w");

   fprintf(f, "\n");
   for (i = (size - 1); i >= 0; i--)
   {
       if (getbit(word, i) == 0) fprintf(f, "0");
       else if (getbit(word, i) == 1) fprintf(f, "1");
       else
       {
30         fprintf(f, " error printing bits ");
           return(FAILURE);
       }
   }

   fprintf(f, "\n");

   return(SUCCESS);
}
35

```

```

/*
 * DISPLAY_BITMAP
 *
 * display_bitmap will display the bitmap bit pattern on the standard output.
 *
 * FORMAT
 *
 * int display_bitmap(struct bitmap *map);
 *
 * ARGUMENTS
 *
 * structure bitmap *map
 *     a pointer to a bitmap structure which will be displayed.
 *
 * RETURN VALUES
 *
 * SUCCESS - the printing was successful.
 * FAILURE - the printing was not successful.
 */

int display_bitmap(map)
struct bitmap *map;
{
    int *current;
    int counter = 0;

    unsigned int *temp;

    if (map == NULL) return(FAILURE);

    temp = map->start;

    while(TRUE)
    {
        if(temp < map->end)
        {
            print_bits(*temp,BITMAP_INTEGER_SIZE);
            temp++;
        }
        if(temp < map->end)
        {
            print_bits(*temp,BITMAP_INTEGER_SIZE);
            temp++;
        }
        else
        {
            print_bits(*map->end,map->end_bits);
            printf("\n");
            return(SUCCESS);
        }
        printf("\n");
    }
}

```

```

5      /
      == WRITE_BITMAP
      ==
      == write_bitmap will write the bitmap bit pattern to a file.
      ==
      == FORMAT
      ==
      == int write_bitmap(struct bitmap *map);
      ==
      == ARGUMENTS
      ==
10     == structure bitmap *map
      ==     a pointer to a bitmap structure which will be displayed.
      ==
      == RETURN VALUES
      ==
      == SUCCESS - the writing was successful.
      == FAILURE - the writing was not successful.
      ==
15     int write_bitmap(map, filename)
      struct bitmap *map;
      char *filename;
      {
          int *current;
          int counter = 0;
          unsigned int *temp;

20         if (map == NULL)
            return(FAILURE);

          temp = map->start;

          while(TRUE)
          {
              if(temp < map->end)
              {
25                 write_bits(filename, temp, BITMAP_INTEGER_SIZE);
                  temp++;
              }
              if(temp < map->end)
              {
                  write_bits(filename, temp, BITMAP_INTEGER_SIZE);
                  temp++;
              }
              else
              {
30                 write_bits(filename, *map->end, map->end_bits);
                  printf("\n");
                  return(SUCCESS);
              }
              printf("\n");
          }
      }

35

```

```

/*
 * LOAD_BITMAPS
 *
 * Invoked by fdc_prototype as part of the load process to read in
 * all of the up to 20 subsidiary bitmap universe definition bitmaps
 *
 * These bitmaps are created by load program and are of the same size as
 * the master file. These bitmaps can be compared directly to any other master
 * bitmap table.
 */

void load_bitmaps()
{
    unsigned int num_items, num_bits, num_alloc;
    char file_name[FILE_NAME_LENGTH + 1];
    int nn, nn1;
    struct bitmap *bitmap_file;
    FILE *ff;
    char label[FIELD_LABEL_LENGTH + 1];

    for ( nn=1; nn<=set_subsidary; nn++ )
    {
        /*
         * construct name and file number.
         */

        sprintf( file_name, "bitmap_dir.sub%02.2d_bitmap.bmp", nn);

        if ((ff = fopen(file_name, "r")) == NULL)
            break;

        /*
         * read the total count
         */

        fread(&num_items, sizeof(num_items), 1, ff);

        /*
         * read and create the subsidiary bitmaps
         */

        if ( (subsidiary_bitmap[nn] = create_bitmap(num_items)) == NULL )
            error_handler (BITMAP_NOT_LOAD, ERROR, NO_STATUS, file_name);

        num_alloc = num_items/BITMAP_INTEGER_SIZE;

        if (num_items % BITMAP_INTEGER_SIZE)
            num_alloc++;

        /*
         * read bitmap now
         */

        fread(subsidiary_bitmap[nn]->start, sizeof(subsidiary_bitmap[nn]), num_alloc, ff);

        /* end of read successfully opened bitmap file */

        /* If a given subsidiary file cannot be open */
        if (ff == NULL)
        {
            error_handler (FILE_NOT_OPEN, ERROR, NO_STATUS, file_name);
        }
    }
}

```

```

/*
 * LOAD_PUR_PRD_MAPS
 *
 * Invoked by fdc_prototype as part of the load process to read in
 * the purchase and product count definition maps
 *
 * These maps are created by load program and are of the same size as
 * the master file.
 */

void load_pur_prd_maps()
{
    unsigned int num_bits, num_alloc;
    char file_name[FILE_NAME_LENGTH + 1];
    struct bitmap *bitmap_file;
    FILE *f;
    char label[FIELD_LABEL_LENGTH + 1];
    unsigned int *ptr;
    unsigned int jj, kk, i, j;
    unsigned short *sptr;
    unsigned int num_blocks;
    int status;

    /* do openmapfile unless set for flushing purchase map data. if delete_pur_info,
     then openmapfile and unmapclose is done as part of purchase query processing */
    if ( set_purchase && !delete_pur_info )
    {

        /* 256 is blocksize 512 divide by 2 bytes per short */
        num_blocks = max_number_of_bits / 256;

        if ( max_number_of_bits % 256 )
            num_blocks++;

        status = OpenMapFile( "bitmap_dir.pur01_map.cnt", "toolbar.sec", "purchase map count",
            &cus_pur_chnl, &cus_pur_addr, 1, num_blocks );

        if (is_error(status))
            error_handler (MAP_OPEN_ERR, ERROR, status, "bitmap_dir.pur01_map.cnt in load_pur_prd_maps");

        pur01_map = cus_pur_addr.start;

        /* 1st 2 records together contain total # of customer records map count data starts after total */
        i = *pur01_map;
        pur01_map++;
        j = *pur01_map;
        i += j*65536; /* to int from 2 short */
        /* this # of customers must match # of customers from master_file or we arent set up
         properly for consistent data base */
        if ( i != max_number_of_bits )
            error_handler (MASTER_MAPCNT_MATCH_FAIL, ERROR, i,
                "cust lg purch map count mismatch in load_pur_prd_maps");

        pur01_map_count = i; /* must set this size */

        /* will point to beginning of customer to purchase map count data */
        pur01_map++;

        /* now create purchase bitmap of all bits to 1 */
        /* size of map is in external max_purchase_bits */
        if ( (pur01_bitmap = create_general_bitmap( max_purchase_bits )) == NULL )
            error_handler (BITMAP_NOT_LOAD, ERROR, NO_STATUS, file_name);
        for ( ptr=pur01_bitmap->start; ptr<pur01_bitmap->end; ptr++ )
            *ptr = -0;
    }
}

```

```

        jj = 1;
        for ( i=0, ptr=pur01_bitmap->end; i<pur01_bitmap->end_bits; i++)
        {
5          *ptr |= jj;
          jj <<= 1;
        }
    }

    /* now for product file */
    /* do openmapfile unless set for flushing product map data if delete_prd_info,
    then openmapfile and unmapclose is done as part of product query processing */
    if ( set_product && !delete_prd_info )
    {
10      /* 256 is blocksize 512 divide by 2 bytes per short */
      num_blocks = max_number_of_bits / 256;

      if ( max_number_of_bits % 256 )
        num_blocks++;

      status = OpenMapFile( "bitmap_dir:prd01_map.cnt", "foobar.sec", "product map count",
        &cus_prd_cntrl, &cus_prd_addr, 1, num_blocks );

15      if (is_error(status))
        error_handler (MAP_OPEN_ERR, ERROR, status, "bitmap_dir:prd01_map.cnt in load_pur_prd_maps");

      prd01_map = cus_prd_addr.start;

      /* 1st 2 records together contain total # of customer records. map count data starts after total */
      i = *prd01_map;
      prd01_map++;
      j = *prd01_map;
      i += j*65536; /* to int from 2 short */
20      /* this # of customers must match # of customers from master_file or we aren't set up
      properly for consistent data base. */
      if ( i != max_number_of_bits )
        error_handler (MASTER_MAPCNT_MATCH_FAIL, ERROR, i,
          "cust to purch map count mismatch in load_pur_prd_maps");

      /* will point to beginning of customer to purchase map count data */
      prd01_map++;

25      prd01_map_count = i; /* must set this size */

      /* now create product bitmap of all bits to 1 */
      if ( (prd01_bitmap = create_general_bitmap( max_product_bits )) == NULL )
        error_handler (BITMAP_NOT_LOAD, ERROR, NO_STATUS, "prd01_bitmap in load_pur_prd_maps");
      for ( ptr=prd01_bitmap->start; ptr<prd01_bitmap->end; ptr++)
        *ptr = -0;

      jj = 1;
      for ( i=0, ptr=prd01_bitmap->end; i<prd01_bitmap->end_bits; i++)
30      {
        *ptr |= jj;
        jj <<= 1;
      }
    }

    /* now get purchase to product map count */
    if ( set_purchase && set_product && !delete_pur_prd_info )
    {
35      /* one product count record for each purchase */
      /* 256 is blocksize 512 divide by 2 bytes per short */
      num_blocks = max_purchase_bits / 256;

```

```

    if ( max_purchase_bits % 256 )
        num_blocks++;

    status = OpenMapFile( "bitmap_dir.pur_prd01_map.cnt", "toobar.sec", "purchase product map count",
        &pur_prd_chnl, &pur_prd_addr, 1, num_blocks );

5    if ( is_error(status) )
        error_handler (MAP_OPEN_ERR, ERROR, status,
            "bitmap_dir.pur_prd01_map.cnt from load_pur_prd_maps");

    pur_prd01_map = pur_prd_addr.start;

    i = *pur_prd01_map;
    pur_prd01_map++;
    j = *pur_prd01_map;
10    i += j*65536; /* to int from 2 short */
    /* this # of purchases must match # of purchases from master_file or we arent set up
    properly for consistent data base. */
    if ( i != max_purchase_bits )
        error_handler (MASTER_MAPCNT_MATCH_FAIL, ERROR, i,
            "purch to product map count not matched in load_pur_prd_maps");

    pur_prd01_map_count = i; /* must set this size */

    /* will point to beginning of customer to purchase map count data */
15    pur_prd01_map++;

    /* size of map is in external max_purchase_bits */
    if ( (pur_prd01_bitmap = create_general_bitmap( max_purchase_bits )) == NULL )
        error_handler (BITMAP_NOT_LOAD, ERROR, NO_STATUS, "pur_prd01_bitmap in load_pur_prd_maps");
    jj = 1; kk = BITMAP_INTEGER_SIZE;
    for ( ptr=pur_prd01_bitmap->start, sptr = pur_prd01_map, i=0; i<max_purchase_bits; sptr++, i++)
    {
        if ( *sptr )
20        {
            *ptr |= jj;
        }
        if ( --kk )
            jj <<= 1;
        else
        {
            kk = BITMAP_INTEGER_SIZE;
            jj = 1;
            ptr++;
25        }
    }

    /* END load_pur_prod_maps */

    /*
    -- SET_COUNTER
    -- This routine is called once to initialize bit_counts array
30    -- with the count of bits that are set in integers 1,2,...,255
    -- (ie, 0 has 0, 1 has 1, 2 has 1, 3 has 2, ... 255 has 8
    -- bit_counts is used in a fast count set bits algorithm
    */

    set_counter()

35    {
        int i,j,one;

```



```

    for(i=0;i<256;i++)
    {
        bit_counts[i] = 0;
        one = 1;
        for(j=0;j<8;j++)
        {
            if(one & i) bit_counts[i] += 1;
            one = one<<1;
        }
    }
}

/*
10  -- MODULE: SEARCH_SECTION.C
    --
    -- MODULE DESCRIPTION:
    --
    -- Set of routines which manipulate global sections to be used
    -- as datasets for variables.
    --
    -- AUTHORS:
    --
15  --      Kelly Westman      Digital Equipment Corporation
    --      Sushil Pillai     Digital Equipment Corporation
    --
    -- CREATION DATE: 3-August-1992
    --
    -- DESIGN ISSUES:
    --
    --      Performs a linear search through a dataset. Does not do any sorting
    --      or presupposes a sorted order to data.
    --      Search routines depend on datatype of field being searched - avoided
20  --      function calls in middle of loop for speed
    --
    --
    -- PORTABILITY ISSUES.
    --
    --      Uses VMS system calls for create, mapping and unmapping global sections
    --      Uses RMS attributes blocks for file to open
    --
    -- MODIFICATION HISTORY:
    --
25  --      3-August-1992 - Original
    --      18-May-1993  - Chuck Malmeskog
    --          - Replace fail_and_exit calls with error_handler() routine
    --      24-Aug-1993  - Chuck Malmeskog
    --          - Added greater_than_equal switch code for search_large_field_array()
    --      30-Sep-1993  - Charles Malmeskog
    --          - Update error_handler() calls to include ERROR message type
    --
    --
30  --
    --
    -- INCLUDE FILES
    --
    --
    --#include <stdio.h>
    --#include <stdlib.h>
    --#include <ssdef.h>
    --#include <rms.h>
    --#include <fab.h>
    --#include <secddef.h>
    --#include <psldef.h>
    --#include <descrip.h>
    --#include <unixio>
    --#include <file>
    --#include <math>
35

```

```

/*include "tdc_parser"*/
#include "tdc_prototype"
#include "tdc_macro_defn"
5 #include "tdc_error_numbers.h"

/* global declarations */
extern enum pur_prd_query_type query_type;

int cus_pur_chnl;          /* OPENMAPFILE channel for customer to purchase map count data */
int cus_prd_chnl;          /* OPENMAPFILE channel for customer to product map count data */
int pur_prd_chnl;          /* OPENMAPFILE channel for purchase to product map count data */

10 struct address_range cus_pur_addr; /* addresses of mapped customer to purchase map count data for OPENMAPFILE.
UNMAPCLOSE
struct address_range cus_prd_addr; /* addresses of mapped customer to product map count data for OPENMAPFILE.
UNMAPCLOSE */
struct address_range pur_prd_addr; /* addresses of mapped purchase to product map count data for OPENMAPFILE.
UNMAPCLOSE */

/* flush flags. mark if map count data is mapped or not. Always 0, unless corresponding delete_pur_info flag is on, for
allowing dynamic map and unmap. (if delete_pur_info is 0, then map counts always mapped and these flag stay at 0 */
int cus_pur_map = 0;        /* flush flags. 1=mapped. 0=not mapped. customer to purchase map count data */
int cus_prd_map = 0;        /* flush flags. 1=mapped. 0=not mapped. customer to purchase map count data */
15 int pur_prd_map = 0;      /* flush flags. 1=mapped. 0=not mapped. customer to purchase map count data */

/* external declarations */
/* structures for fixed strings */
extern int purchase_query;
extern int product_query;

extern struct fixed_string_type fixed_field[NUM_FIXED_STRINGS];
extern int max_number_of_bits;

20 /* purchase variable, product variables */
extern int set_purchase;
extern int set_product;

extern unsigned int delete_pur_info;
extern unsigned int delete_prd_info;
extern unsigned int delete_pur_prd_info;

/* structures for bitmaps */
extern struct bitmap *subsidiary_bitmap[SUB_FILE_MAX];
25 extern struct bitmap *pur01_bitmap;
extern unsigned short *pur01_map;

extern unsigned int pur01_map_count;

extern struct bitmap *prd01_bitmap;
extern unsigned short *prd01_map;
extern unsigned int prd01_map_count;

30 extern unsigned int pur_prd01_map_count;
extern unsigned short *pur_prd01_map;

extern int max_number_of_bits;
extern int max_purchase_bits;

extern int DMO_CONNECTED;

/* This module contains the following routines:

35 int UnmapCloseFile(chan, adr);
int OpenMapFile(fname, dname, section_name, chan, retadr, vbn, numblock);
void explode_bitmaps (in_query_bitmap, results_bitmap, sub_defn_bitmap, number_of_items);
int search_for_value( field,field1, operations, operations1, results_bitmap);
int search_large_array_for_value(array_start, compare_value, next_compare_value,
compare_list_value,operator, num_bits, results_bitmap);

```

```

int search_medium_array_for_value(array_start, compare_value, next_compare_value,
                                compare_list_value, operator, num_bits, results_bitmap);
int search_short_array_for_value(array_start, compare_value, next_compare_value,
                                compare_list_value, operator, num_bits, results_bitmap);
int search_float_array_for_value(array_start, compare_value, next_compare_value,
                                compare_list_value, operator, num_bits, results_bitmap);
int search_double_array_for_value(array_start, compare_value, next_compare_value,
                                compare_list_value, operator, num_bits, results_bitmap);
5 int search_bit_array_for_value(array_start, operator, num_bits, results_bitmap);
int search_string_array_for_value(array_start, compare_value, length,
                                search_value, operator, num_bits, results_bitmap);
int search_fstring_array_for_value(array_start, compare_str, length,
                                field_offset, operator, num_bits, results_bitmap);
int search_mixed_string_for_value(array_start, compare_value, length,
                                search_value, operator, num_bits, results_bitmap);
int search_mixed_fstring_for_value(array_start, compare_str, length, field_offset,
                                search_value, operator, num_bits, results_bitmap);
10 int search_large_field_array(array_start, array_start1, operator, num_bits, results_bitmap);
int search_medium_field_array(array_start, array_start1, operator, num_bits, results_bitmap);
int search_short_field_array(array_start, array_start1, operator, num_bits, results_bitmap);
int search_double_field_array(array_start, array_start1, operator, num_bits, results_bitmap);
int search_float_field_array(array_start, array_start1, operator, num_bits, results_bitmap);
int search_bit_field_array(array_start, array_start1, operator, num_bits, results_bitmap);
int search_string_field_array(array_start, array_start1, operator, num_bits,
                             length, results_bitmap);
int search_fstring_field_array(array_start, array_start1, operator,
                             f_offset, f_offset1, input_bitmap, results_bitmap);
15 int check_list_for_mixed_strings( search_list );
int search_dataset_for_value( data, data1, field_type, field_size, field_offset,
                             field_offset1, operator, search_value,
                             search_value_type, results_bitmap);
int search_mf_large_field_array(array_start, array_start1, operator,
                                input_bitmap, input_bitmap1, results_bitmap);
int search_mf_medium_field_array(array_start, array_start1, operator,
                                input_bitmap, input_bitmap1, results_bitmap);
20 int search_mf_short_field_array(array_start, array_start1, operator,
                                input_bitmap, input_bitmap1, results_bitmap);
int search_mf_bit_field_array(array_start, array_start1, operator,
                                input_bitmap, input_bitmap1, results_bitmap);
int search_mf_string_field_array(array_start, array_start1, operator, length,
                                input_bitmap, input_bitmap1, results_bitmap);
int search_mf_fstring_field_array(array_start, array_start1, operator, length, f_offset,
                                f_offset1, input_bitmap, input_bitmap1, results_bitmap);
int search_mf_pp_large_field_array(array_start, array_start1, operator, input_bitmap,
                                input_bitmap1, results_bitmap, map_count, map_count1,
                                map_count2, pp_oper_code, pp_oper_code1, sub_flag);
25 int search_mf_pp_medium_field_array( array_start, array_start1, operator, input_bitmap,
                                input_bitmap1, results_bitmap, map_count, map_count1,
                                map_count2, pp_oper_code, pp_oper_code1, sub_flag );
int search_mf_pp_short_field_array( array_start, array_start1, operator, input_bitmap,
                                input_bitmap1, results_bitmap, map_count, map_count1,
                                map_count2, pp_oper_code, pp_oper_code1, sub_flag );
int search_mf_pp_string_field_array( array_start, array_start1, operator, length,
                                input_bitmap, input_bitmap1, results_bitmap,
                                map_count, map_count1, map_count2,
                                pp_oper_code, pp_oper_code1, sub_flag );
30 int search_mf_pp_fstring_fieldarray( array_start, array_start1, operator, length, f_offset,
                                f_offset1, input_bitmap, input_bitmap1, results_bitmap,
                                map_count, map_count1, map_count2,
                                pp_oper_code, pp_oper_code1, sub_flag );
int search_mf_dataset_arrays( data, data1, field_type, field_size, field_offset, field_offset1,
                                operator, search_value, search_value_type, input_bitmap,
                                input_bitmap1, results_bitmap);
int search_mf_pp_dataset_arrays( data, data1, field_type, field_size, field_offset, field_offset1,
                                operator, search_value, search_value_type, input_bitmap, input_bitmap1,
                                results_bitmap, map_count, map_count1, map_count2, oper_code,
                                oper_code1, sub_flag );
35

```

*/

/* if we go to named global sections */

/* static \$DESCRIPTOR(global_section_name, "CUSTOMER"); */

```

1  /
2  --
3  -- UNMAPCLOSEFILE
4  --
5  --      Unmap and close a previously mapped file.
6  --
7  -- FORMAT
8  --
9  --      int UnmapCloseFile(int chan, struct address_range *adr)
10 --
11 -- ARGUMENTS
12 --
13 --      chan - i/o channel assigned to the file, (input)
14 --      adr - Starting address where the file is mapped, (input)
15 --
16 -- RETURNS
17 --
18 --      Status value.
19 --
20 /

int UnmapCloseFile(chan, adr)
int
struct address_range *adr,      chan;
{
15     int                      status;

    status = sys$purgev(adr, NULL, NULL);
    if (is_error(status)) return (status);

    status = sys$dehva(adr, NULL, NULL);
    if (is_error(status)) return (status);

    status = sys$dassgn(chan);
    if (is_error(status)) return (status);
20     return (SSS_NORMAL);
}

/
-- OPENMAPFILE
--
-- NOTE: OPENMAPFILE - for reading only, use WRITEMAPFILE for write access
--
25 --      Create file and map to the virtual address space.
--
-- FORMAT:
--
--      int OpenMapFile(fname, dfname, chan, retadr, vbn, numblock)
--
--      char *fname;
--      char *dfname;
--      int *chan;
--      struct address_range *retadr;
30 --      int vbn;
--      int numblock;
--
-- ARGUMENTS:
--
--      fname - input file name, (input)
--      dfname - default file name, (input)
--      chan - i/o channel assigned, (output)
--      retadr - Return address where the file is mapped, (output)
--      vbn - virtual block number of data start
35 --      numblock - number of data blocks

```

```

    RETURNS:
    Status value.
*/

5  int OpenMapFile(fname, dfname, section_name, chan, reladr, vbn, numblock)
    char *fname;
    char *dfname;
    char *section_name;
    int *chan;
    struct address_range *reladr;
    int vbn;
    int numblock;
{
10  int status;
    struct address_range in_adr;
    struct FAB fab;
    struct XABFHC xab;
    /* int sec_flags=SECSM_GBLISECSM_SYSGBLISECSM_EXPREG; */
    int sec_flags=SECSM_EXPREG;
    int access_mode=PSLSC_USER;
    char *buffer;
    SDESCRIPTOR(global_section_name, section_name);

15  xab = cc$rms_xabfhc; /* fill it with defaults */

    fab = cc$rms_fab; /* fill it with defaults */
    fab.fab$l_dna = dfname;
    fab.fab$b_dns = strlen(dfname);
    fab.fab$l_fna = fname;
    fab.fab$b_fns = strlen(fname);
    /* fab.fab$b_sbr = FABSM_MSE; /* get access */
    fab.fab$b_fac = FABSM_BRO; /* get access */
    /* fab.fab$b_sbr = FABSV_UPI; */
    fab.fab$l_fop = FABSM_UFO; /* no rms */
    fab.fab$l_xab = &xab; /* extended attribute block */

    status = sys$open(&fab);
    if (is_error(status)) return (status);

    *chan = fab.fab$l_stv;

25  /*
    if (xab.xab$w_ffb == 0)
        numblock = xab.xab$l_ebk - 1;
    else
        numblock = xab.xab$l_ebk;
    */

    in_adr.start = 0x200; /* any program region address */
    in_adr.end = 0x200; /* ditto, can be same as start */

30  status = sys$cmposc(&in_adr, /* requested addresses */
    reladr, /* addresses mapped */
    NULL, /* Access Mode */
    sec_flags, /* map first available space */
    NULL, /* ident */
    NULL, /* global section name */
    NULL, /* page to start mapping */
    *chan, /* channel */
    numblock, /* page count */
    vbn, /* virtual block number */
    NULL, /* protection */
    NULL);

35  if (is_error(status)) return (status);

```

```

    return (SSS_NORMAL);
}

5  /*
   * WRITEMAPFILE
   *
   * NOTE: WRITEMAPFILE - for write access, use OPENMAPFILE if read access only is needed
   *
   * Create file and map to the virtual address space.
   *
   * FORMAT:
   *
   * int WriteMapFile(fname, dfname, chan, retadr, vbn, numblock)
10  /*
   *     char *fname;
   *     char *dfname;
   *     int *chan;
   *     struct address_range *retadr;
   *     int vbn;
   *     int numblock;
   *
   * ARGUMENTS:
15  /*
   *     fname - input file name, (input)
   *     dfname - default file name, (input)
   *     chan - i/o channel assigned, (output)
   *     retadr - Return address where the file is mapped, (output)
   *     vbn - virtual block number of data start
   *     numblock - number of data blocks
   *
   * RETURNS:
20  /*
   *     Status value.
   */

int WriteMapFile(fname, dfname, section_name, chan, retadr, vbn, numblock)
char *fname;
char *dfname;
char *section_name;
int *chan;
25 struct address_range *retadr;
int vbn;
int numblock;
{
    int status;
    struct address_range in_addr;
    struct FAB fab;
    struct XABFHC xab;
    int sec_flags = SECSM_EXPREG
30 | SECSM_WRT;

    xab = cc3rms_xabfhc;
    fab = cc3rms_fab;
    fab.fab$1_dna = dfname;
    fab.fab$b_dns = strlen(dfname);
    fab.fab$1_fna = fname;
    fab.fab$b_fns = strlen(fname);
    fab.fab$b_fac = FABSM_PUT
35 | FABSM_GET;
    fab.fab$b_sfr = FABSM_UPI;
    fab.fab$b_sfr = FABSM_SHRPUT
    | FABSM_SHRGET;
}

```

```

1  fab.fab$I_top = FAB$M_UFO;

2  status = sys$open(&fab);
3  if (is_error(status)) return (status);

5  *chan = fab.fab$I_stv;

6  in_adr.start = 0x200;
7  in_adr.end = 0x200;

8
9
10 status = sys$crmpsc(&in_adr,           /* requested addresses */
11                    retadr,             /* addresses mapped */
12                    NULL,                /* Access Mode */
13                    sec_flags,           /* map first available space */
14                    NULL,                /* global section name */
15                    NULL,                /* ident */
16                    NULL,                /* page to start mapping */
17                    *chan,               /* channel */
18                    numblock,            /* page count */
19                    vbn,                 /* virtual block number */
20                    NULL,                /* protection */
21                    NULL);

22 if (is_error(status)) return (status);

23 return (SS$NORMAL);
24 )

25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

234

```

/* j indices are bit flags 1,2,4,8,16,32. (hex - 1,2,4,8,10,20,40,...80000000 used to check bitmap table
   bit by bit. kk indices are counts starting at 32 and counting down to 0, to tell when to move to the next
   word in the data tables */

5  in_bitmap = in_query_bitmap->start; /* points to subsidiary query results bitmap of size of subsid */
   out_bitmap = results_bitmap->start; /* bitmap we are creating - of size of master_file all bits 0 on entry */
   sub_def_map = sub_defn_bitmap->start; /* master_file size bitmap defining elements in subsidiary file */

   kk_out_map = kk_in_map = BITMAP_INTEGER_SIZE;
   ll_out_map = ll_in_map = 1;

   for ( i=0; i<number_of_items; i++)
   {
       /* if this data item is in the sub_defn_file */
       if ( *sub_def_map & ll_out_map )
       {
           /* see if set in sub_results query map */
           if ( *in_bitmap & ll_in_map )
           {
               *out_bitmap |= ll_out_map; /* yes, set in resultant bitmap */
           }
           /* bump to next item in query_results map */
           if ( --kk_in_map )
               ll_in_map <<= 1;
15         else
         {
             in_bitmap++;
             ll_in_map = 1;
             kk_in_map = BITMAP_INTEGER_SIZE;
         }
       }
       /* bump pointer to next bit in subsidiary_defn bitmap and results bitmap, and
          advance to next word of bits if necessary */
       if ( --kk_out_map )
           ll_out_map <<= 1;
20         else
         {
             out_bitmap++;
             sub_def_map++;
             ll_out_map = 1;
             kk_out_map = BITMAP_INTEGER_SIZE;
         }
       }
   }

25 )

/*
-- GET_BITMAP(field_name, num_bits)
-- Used for multifile processing to select proper subsidiary (subsidiary, purchase,
-- or product) file bit definition file.
--
-- Each sub_file has been read in as part of the load, is of same length as
30 -- the master file, and has bits set marking elements in the sub_file.
--
-- field_name is ascii string in the form of
--     - SUB01,02,...,20 - subsidiary file
--     - PUR               - Purchase file
--     - PRD               - Product file
--
-- Note: if the named sub_file doesn't exist or wasn't loaded, will return a NULL
35 */

```



```

struct bitmap *get_bitmap(field_name, num_bits)
char          field_name;
int           num_bits;
{
    int temp_bit_select;
    struct bitmap *output_bitmap;
    unsigned int i, j=1;

    if (strcmp(field_name, "CUS", 3) == 0)
    {
        if ((output_bitmap = create_bitmap(num_bits)) == NULL)
            error_handler (BITMAP_NOT_CREATE, ERROR, NO_STATUS, "output bitmap in get_bitmap");

        /* set all bits of my output_bitmap to 1 */
        for( temp = output_bitmap->start; temp < output_bitmap->end ; temp++)
            temp = -0;

        for( temp = output_bitmap->end; i=0; i<output_bitmap->end_bits; i++ )
        {
            temp |= j;
            j <<= 1;
        }

        return(output_bitmap);
    }

    else if (strcmp(field_name, "SUB", 3) == 0)
    {
        sscanf(field_name+3, "%d", &bit_select);
        return(subsidiary_bitmap[bit_select]);
    }

    else if (strcmp(field_name, "PUR", 3) == 0)
        return(pur01_bitmap);

    else if (strcmp(field_name, "PRD", 3) == 0)
        return(prd01_bitmap);
}

/* SEARCH_FOR_VALUE */

int search_for_value (struct field_entry *field, struct field_entry *field1,
                     struct field_operations *operations,
                     struct field_operations *operations2,
                     struct bitmap *results_bitmap);

int search_for_value( field,
                      field1,
                      operations,
                      operations1,
                      results_bitmap)
struct field_entry *field;
struct field_entry *field1;
struct field_operations *operations;
struct field_operations *operations1;
struct bitmap *results_bitmap;
{

```

```

    int
    int
    int
    int
    struct address_range retadr;
    struct address_range retadr1;
    int
    int
    struct bitmap
    unsigned int
    int
    int
    int
    unsigned short
    unsigned int
    int

    status;
    chan;
    chan1;
    total_count = 0;

    i, field_offset=0, field_offset1=0;
    bit_select;
    *input_bitmap, *input_bitmap1;
    *temp;
    reference_count;
    subsidiary;
    num_blocks;
    *pp_map=NULL, *pp_map1=NULL, *pp_map2=NULL,
    op_code=0, op_code1=0;
    sub_flag=0;

    status = OpenMapFile(field->table, "foobar.sec", field->field_name, &chan,
        &retadr, field->vbn, field->number_of_blocks);
    if (is_error(status))
        error_handler (MAP_OPEN_ERR, ERROR, status, "First field in search_for_value");

    if ( (strcmp(field->field_name, "PUR", 3) == 0) ||
        (field1 != NULL && (strcmp(field1->field_name, "PUR", 3) == 0)) )
    {
        /* if flushing delete flags set, deal with mapping various purchase/product map count data */
        if ( set_purchase && delete_pur_info && !cus_pur_map )
        {
            /* 256 is blocksize 512 divide by 2 bytes per short */
            num_blocks = max_number_of_bits / 256;

            if ( max_number_of_bits % 256 )
                num_blocks++;

            status = OpenMapFile( "bitmap_dir/pur01_map.cnt", "foobar.sec", "purchase map count",
                &cus_pur_chan1, &cus_pur_addr, 1, num_blocks );

            if (is_error(status))
                error_handler (MAP_OPEN_ERR, ERROR, status, "cus-pur-map count data in search_for_value");

            pur01_map = cus_pur_addr.start;

            pur01_map += 2; /* bump past count of records to 1st customer count data */

            cus_pur_map = 1; /* mark customer to purchase map count data mapped */
        }
    }

    if ( (strcmp(field->field_name, "PRD", 3) == 0) ||
        (field1 != NULL && (strcmp(field1->field_name, "PRD", 3) == 0)) )
    {
        if ( set_product && delete_prd_info && !cus_prd_map )
        {
            /* 256 is blocksize 512 divide by 2 bytes per short */
            /* uses # of customers */
            num_blocks = max_number_of_bits / 256;

            if ( max_number_of_bits % 256 )
                num_blocks++;

            status = OpenMapFile( "bitmap_dir/prd01_map.cnt", "foobar.sec", "product map count",
                &cus_prd_chan1, &cus_prd_addr, 1, num_blocks );

            if (is_error(status))
                error_handler (MAP_OPEN_ERR, ERROR, status, "cus-prd-map count data in search_for_value");

            prd01_map = cus_prd_addr.start;
        }
    }

```

```

        prd01_map += 2;      /* bump past count of records to 1st customer count data */
        cus_prd_map = 1;     /* mark customer to purchase map count data mapped */
    }
5   }

    /* if both PUR and PRD then get purchase to product map count */
    if ( (strcmp(field->field_name, "PUR", 3) == 0) || (strcmp(field->field_name, "PRD", 3) == 0) ||
        ((field1 != NULL) &&
         ((strcmp(field1->field_name, "PUR", 3) == 0) || (strcmp(field1->field_name, "PRD", 3) == 0))
         &&
         (strcmp(field->field_name, field1->field_name, 3) != NULL) ) ) ) )
    {
10      if ( set_purchase && set_product && delete_pur_prd_info && !pur_prd_map )
      {

          /* 256 is blocksize 512 divide by 2 bytes per short */
          /* uses # of purchase */
          num_blocks = max_purchase_bits / 256;

          if ( max_purchase_bits % 256 )
              num_blocks++;

15      status = OpenMapFile( "bmap_dir.purprd_map.ent", "foobar.sec", "purchase product map count",
                           &pur_prd_chan1, &pur_prd_addr, 1, num_blocks );

          if ( is_error(status) )
              error_handler (MAP_OPEN_ERR, ERROR, status, "pur-prd-map count data in search_for_value");

          pur_prd01_map = pur_prd_addr.start;

          pur_prd01_map += 2; /* bump past count of records to 1st customer count data */

20      pur_prd_map = 1;     /* mark customer to purchase map count data mapped */
      }
    }

    field->data->items = retadr.start;

    if (field->field_type == fixed_string)
25      field_offset = get_fixed_string_offset(field->field_name);

    /*
     * field comparison rather than value comparison
     */

    if (field1 != NULL)
    {
30      status = OpenMapFile(field1->table, "foobar.sec", field1->field_name,
                           &chan1, &retadr1, field1->vbn, field1->number_of_blocks);

          if ( is_error(status) )
              error_handler (MAP_OPEN_ERR, ERROR, status, "search_for_value, second field");

          field1->data->items = retadr1.start;

          if (field1->field_type == fixed_string)
35      field_offset1 = get_fixed_string_offset(field1->field_name);
    }

```

```

    }
    else
    {
        if (strcmp(field->field_name, "PUR", 3) == 0)
            pp_map = pur01_map;
        else
            if (strcmp(field->field_name, "PRD", 3) == 0)
                pp_map = prd01_map;
        if ((strcmp(field->field_name, "PUR", 3) == 0) ||
            (strcmp(field->field_name, "PRD", 3) == 0))
            sscanf(field->field_name+3, "%d", &op_code);

        if (strcmp(field1->field_name, "PUR", 3) == 0)
            pp_map1 = pur01_map;
        else
            if (strcmp(field1->field_name, "PRD", 3) == 0)
                pp_map1 = prd01_map;
        if (pp_map && pp_map1 && pp_map != pp_map1)
            pp_map2 = pur_prd01_map;
        if ((strcmp(field1->field_name, "PUR", 3) == 0) ||
            (strcmp(field1->field_name, "PRD", 3) == 0))
            sscanf(field1->field_name+3, "%d", &op_code1);

        if ((strcmp(field->field_name, "SUB", 3) == 0) ||
            (strcmp(field1->field_name, "SUB", 3) == 0))
            sub_flag++;

        /* Case1: Check to see if either of the field is not a CUS */
        /* and thus a multi_file type search is needed. */
        if ((strcmp(field->field_name, "CUS", 3) != 0) ||
            (strcmp(field1->field_name, "CUS", 3) != 0))
        {
            /* process the query, based on the pre-defined bitmaps */
            /* of the both fields */

            input_bitmap = get_bitmap(field->field_name, results_bitmap->number_of_bits);
            input_bitmap1 = get_bitmap(field1->field_name, results_bitmap->number_of_bits);

            /* If either input_bitmap is NULL, then query specified
            subsidiary table that is not present or no master_bitmap
            set yet. return error */

            if ((input_bitmap == NULL) || (input_bitmap1 == NULL))
            {
                error_handler(NULL_POINTER_ERR, ERROR, NO_STATUS, "input_bitmap in search_for_value");
            }

            if (((strcmp(field->field_name, "CUS", 3) == 0) || (strcmp(field->field_name, "SUB", 3) == 0))
                && ((strcmp(field1->field_name, "CUS", 3) == 0) || (strcmp(field1->field_name, "SUB", 3) == 0)))
            {
                /* cus - sub and sub - sub cases */
                total_count = search_mf_dataset_arrays( field->data,
                                                            field1->data,
                                                            field->field_type,
                                                            field->field_end,
                                                            field_offset,
                                                            field_offset1,
                                                            operations->operator,
                                                            operations->search_value,
                                                            operations->search_value_type,
                                                            input_bitmap,
                                                            input_bitmap1,
                                                            results_bitmap);
            }
        }
    }
}

```

```

        total_count = search_ml_pp_dataset_arrays( field->data,
                                                    field1->data,
                                                    field->field_type,
                                                    field->field_end,
                                                    field_offset,
                                                    field_offset1,
                                                    operations->operator,
                                                    operations->search_value,
                                                    operations->search_value_type,
                                                    input_bitmap,
                                                    input_bitmap1,
                                                    results_bitmap,
                                                    pp_map,
                                                    pp_map1,
                                                    pp_map2,
                                                    op_code,
                                                    op_code1,
                                                    sub_flag);
    }

    if (strcmp(field->field_name, "CUS", 3) == 0)
        free_bitmap(input_bitmap);

15    if (strcmp(field1->field_name, "CUS", 3) == 0)
        free_bitmap(input_bitmap1);
    }
    else
        /* Only possibility is a CUS and CUS case */
        {
            /* Case 3: CUS and CUS case - no multifiles */

            total_count = search_dataset_for_value(field->data,
                                                    field1->data,
20            field->field_type,
            field->field_end,
            field_offset,
            field_offset1,
            operations->operator,
            operations->search_value,
            operations->search_value_type,
            results_bitmap);
        }

25    }

    /*
     * end of field comparisons
     * start of value comparisons
     */

    else
    {
30        if ((strcmp(field->field_name, "CUS", 3)) == 0)
        {
            total_count = search_dataset_for_value(field->data,
                                                    NULL,
                                                    field->field_type,
                                                    field->field_end,
                                                    field_offset,
                                                    field_offset1,
                                                    operations->operator,
                                                    operations->search_value,
                                                    operations->search_value_type,
                                                    results_bitmap);
35        }

        else if (purchase_query)
        {
            /* search_dataset doesnt do average and total cases */

```

```

    if ( (query_type != avg_pur_prd) && (query_type != total_pur_prd) )
    {
        total_count = search_dataset_for_value(field->data,
        5          NULL,
          field->field_type,
          field->field_end,
          field_offset,
          field_offset1,
          operations->operator,
          operations->search_value,
          operations->search_value_type,
          results_bitmap);

        total_count = count_set_bits ( results_bitmap );
    10    }
    else
    {
        total_count = pur_prd_query_special(field->data,
          field->field_type,
          field->field_end,
          field_offset,
          operations->operator,
          operations->search_value,
          operations->search_value_type,
          query_type,
          results_bitmap);
    15    }
  }
  else if (product_query)
  {
    /* search_dataset doesn't do average and total cases */
    if ( (query_type != avg_pur_prd) && (query_type != total_pur_prd) )
    {
    20      total_count = search_dataset_for_value(field->data,
        NULL,
        field->field_type,
        field->field_end,
        field_offset,
        field_offset1,
        operations->operator,
        operations->search_value,
        operations->search_value_type,
        results_bitmap);
    25      total_count = count_set_bits ( results_bitmap );
    }
    else
    {
        total_count = pur_prd_query_special(field->data,
          field->field_type,
          field->field_end,
          field_offset,
          operations->operator,
          operations->search_value,
          operations->search_value_type,
          query_type,
          results_bitmap);
    30    }
  }
}

else if ((strcmp(field->field_name, "SUB", 3)) == 0)
35 {

```

```

total_count = search_dataset_for_value(field->data,
NULL,
field->field_type,
field->field_end,
field_offset,
5  field_offset1,
operations->operator,
operations->search_value,
operations->search_value_type,
results_bitmap);

/* determine if the bitmap needed to be adjusted for multi-file processing */
if (field->data->number_of_items < max_number_of_bits)
10 {
    /* call the explosion routine to set the results_bitmap to the */
    /* "consistent" size. "consistent" size is the size of the */
    /* customer data record file. */

    /* Since the query tree routine, expects the results */
    /* bitmap, I am doing the following: */
    /* - copying the results bitmap to the input bitmap, which */
    /* is now my results bitmap. */
    /* - initializing the results bitmap */
    /* - calling the explosion routine which will explode my */
    /* input bitmap to the appropriate size and store the new */
    /* results in the results bitmap. */
15
    if ((input_bitmap = create_bitmap(results_bitmap->number_of_bits)) == NULL)
        error_handler (BITMAP_NOT_CREATE, ERROR, NO_STATUS, "input bitmap in search_for_value");

    if (copy_bitmaps(input_bitmap, results_bitmap) == NULL)
        error_handler (BITMAP_NOT_COPY, ERROR, NO_STATUS,
20 "results_bitmap to input bitmap in search_for_value");

    for ( temp=results_bitmap->start; temp<=results_bitmap->end; temp++ )
        temp = 0;

    /* explode the value results_bitmap so full size map is returned */

    sscanf((field->field_name+3, "%d", &subsidiary);
    explode_bitmaps (input_bitmap, results_bitmap, subsidiary_bitmap[subsidiary],
25 max_number_of_bits);

    free_bitmap(input_bitmap);
}
}
/* end of value comparisons */

status = UnmapCloseFile(chan, &retadr);
30 if (is_error(status))
    error_handler (UNMAP_CLOSE_ERR, ERROR, status, "search_for_value first field");
if (field1 != NULL)
{
    status = UnmapCloseFile(chan1, &retadr1);
    if (is_error(status))
        error_handler (UNMAP_CLOSE_ERR, ERROR, status, "search_for_value second field");
}

35 /* we do have to free the input bitmap */
return(total_count);
}

```

```

/*
-- SEARCH_LARGE_ARRAY_FOR_VALUE
-- Searches an array of large integer for the given value and updates the bitmap
5 -- Common logic for bitmap table processing with just the operator varying.
-- Search routine macros (ie, SEARCH_FOR ) are in FDC_MACRO_DEFN.H file
-- SEARCH_FOR is invoked as:
--     SEARCH_FOR( oper, missing_flag, missing_value);
-- SEARCH_FOR is passed oper of ==, >=, !=, etc
-- missing flag set to 0 for off and no check needed or 1 to check (> and >= cases
--     need to exclude values set to missing)
-- missing_value is set to specific value for int, medium, short, etc
10 -- SEARCH_FOR macro expands like this:
--
--     kk = BITMAP_INTEGER_SIZE;
--     jj = 1;
--     for ( i=0; i<num_bits; i++)
--     {
--         if (missing_flag)
--         {
15 --             if (( *array_start oper compare_value ) &&
--                 ( *array_start < missing_value ))
--             {
--                 *temp |= jj;
--                 total_count++;
--             }
--         }
--         else
--         {
20 --             if ( *array_start oper compare_value )
--             {
--                 *temp |= jj;
--                 total_count++;
--             }
--             array_start++;
--             if ( --kk )
--                 jj <= 1;
--             else
25 --             {
--                 temp++;
--                 jj = 1;
--                 kk = BITMAP_INTEGER_SIZE;
--             }
--         }
--     }
--
-- SEARCH_FOR_BETWEEN is similar, except for if statement with >= and <= for between
30 */

int search_large_array_for_value(array_start,compare_value,next_compare_value,
                                compare_list_value,operator,num_bits,
                                results_bitmap)

unsigned int    *array_start;
unsigned int    compare_value;
unsigned int    next_compare_value;
struct list_types *compare_list_value;
enum field_operator operator;
35 unsigned int    num_bits;
struct bitmap *results_bitmap;
{

```



```

    unsigned int total_count = 0;
    unsigned int temp;
    unsigned int i, j, jj, kk;
    unsigned int missing_on=1, missing_off=0;
    unsigned int missing_flag=missing_off;
    unsigned int found;

    struct list_types *list_head = NULL;

    list_head = compare_list_value;

    temp = results_bitmap->start;

    switch (operator)
    {
    10     case equal:
            SEARCH_FOR( ==, missing_flag, MISSING_LARGE_VALUE);
            break;
        case equal_list:
            if ( list_head == NULL )
                error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");

            BITMAP_LOOP_PART_1( );
            list_head = compare_list_value;
            while ( list_head != NULL )
            {
                if ( list_head->type_list == variable_value )
                {
                    if ( *array_start == list_head->integer )
                    {
                        temp |= jj;
                        total_count++;
                        break;
                    }
                    /* as our list is sorted low to hi, bail if we know won't find in rest of list */
                    if ( *array_start < list_head->integer )
                        break;
                }
                else
                {
                    if ( list_head->type_list == range_value )
                    {
                        if ( (*array_start >= list_head->between_integer->low) &&
                            (*array_start <= list_head->between_integer->high) )
                        {
                            temp |= jj;
                            total_count++;
                            break;
                        }
                        if ( *array_start < list_head->between_integer->low )
                            break;
                    }
                    list_head = list_head->next;
                }
            }
            array_start++;
            BITMAP_LOOP_PART_2( );
            break;
        case greater_than_equal:
            missing_flag = missing_on;

            /* if value to look for is MISSING_LARGE_VALUE, turn off the */
            /* exclude missing values flag */
            if (compare_value == MISSING_LARGE_VALUE)
            35             missing_flag = missing_off;
    }
}

```

```

SEARCH_FOR( >= , missing_flag, MISSING_LARGE_VALUE);
break;
case greater_than_equal_list:
    if ( list_head == NULL )
5      error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");

    /* set to exclude values set to the large value used for missing */
    missing_flag = missing_on;

    /* case of searching for values greater than or equal a list of values */
    /* simplifies to find the largest value in the list and use */
    /* the normal search routine for this value */

10    compare_value = 0;
    list_head = compare_list_value;
    while ( list_head != NULL )
    {
        if ( list_head->type_list == variable_value )
        {
            if ( list_head->integer > compare_value )
                compare_value = list_head->integer;
        }
        else
15        if ( list_head->type_list == range_value )
        {
            if ( list_head->between_integer->high > compare_value )
                compare_value = list_head->between_integer->high;
        }
        list_head = list_head->next;
    }

    /* if value to look for is MISSING_LARGE_VALUE, turn off the */
    /* exclude missing values flag */
20    if (compare_value == MISSING_LARGE_VALUE)
        missing_flag = missing_off;

    SEARCH_FOR( >= , missing_flag, MISSING_LARGE_VALUE);
    break;
case greater_than:
    missing_flag = missing_on;
    if (compare_value == MISSING_LARGE_VALUE)
        missing_flag = missing_off;
25    SEARCH_FOR( > , missing_flag, MISSING_LARGE_VALUE);
    break;
case less_than_equal:
    SEARCH_FOR( <= , missing_flag, MISSING_LARGE_VALUE);
    break;
case less_than:
    SEARCH_FOR( < , missing_flag, MISSING_LARGE_VALUE);
    break;
case not_equal:
    SEARCH_FOR( != , missing_flag, MISSING_LARGE_VALUE);
30    break;
case not_equal_list:
    if ( list_head == NULL )
        error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");

    BITMAP_LOOP_PART_1();
    list_head = compare_list_value;
    found = 0;
    while ( list_head != NULL )
    {
35        if ( list_head->type_list == variable_value )
        {
            if ( *array_start == list_head->integer )
            {
                found++;
                break;
            }
        }
    }

```

```

        if ( *array_start < list_head->integer )
            break;
    }
    else
        if ( list_head->type_list == range_value )
        {
            if ( (*array_start >= list_head->between_integer->low) &&
                (*array_start <= list_head->between_integer->high) )
            {
                found++;
                break;
            }
            if ( *array_start < list_head->between_integer->low )
                break;
        }
        list_head = list_head->next;
    }

    /* If value not in the list, select it */
    if ( !found )
    {
        *temp |= jj;
        total_count++;
    }

    array_start++;
    BITMAP_LOOP_PART_2( );
    break;

case between:
    SEARCH_FOR_BETWEEN( );
    break;

default:
    error_handler (INVALID_SWITCH_VALUE, ERROR, operator,
        "operator (default) in search_large_array_for_value");
    break;
}
return(total_count);
}

/*
-- SEARCH_MEDIUM_ARRAY_FOR_VALUE
-- Searches an array of medium integer for the given value and updates the
-- bitmap
--
--
--
--
*/
int search_medium_array_for_value(array_start,compare_value,next_compare_value,
                                compare_list_value,operator,num_bits,results_bitmap)
30 unsigned short      *array_start,
    unsigned int      compare_value,
    unsigned int      next_compare_value,
    struct list_types *compare_list_value,
    enum field_operator operator,
    unsigned int      num_bits;
    struct bitmap     *results_bitmap;
{
    unsigned int      total_count = 0;
35 unsigned int      *temp;
    unsigned int      i, j, jj, kk;
    unsigned int      missing_on=1, missing_off=0;
    unsigned int      missing_flag = 0;
    unsigned int      found;

```

246

```

struct list_types *list_head = NULL;

list_head = compare_list_value;

temp = results_bitmap->start;

5  switch (operator)
    {
        case equal:
            SEARCH_FOR( == , missing_off, MISSING_MEDIUM_VALUE);
            break;
        case equal_list:
            if ( list_head == NULL )
                error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");

10         BITMAP_LOOP_PART_1( );
            list_head = compare_list_value;
            while ( list_head != NULL )
            {
                if ( list_head->type_list == variable_value )
                {
                    if ( *array_start == (unsigned short) list_head->integer )
                    {
                        temp |= j;
                        total_count++;
                        break;
                    }
                    if ( *array_start < (unsigned short) list_head->integer )
                        break;
                }
                else

20                 if ( list_head->type_list == range_value )
                {
                    if ( (*array_start >= (unsigned short) list_head->between_integer->low)
                        &&
                        (*array_start <= (unsigned short) list_head->between_integer->high) )
                    {
                        temp |= j;
                        total_count++;
                        break;
                    }
                    if ( *array_start < (unsigned short) list_head->between_integer->low )
                        break;
                }
                list_head = list_head->next;

25             }
            array_start++;
            BITMAP_LOOP_PART_2( );
            break;
        case greater_than_equal:
            SEARCH_FOR( >= , missing_on, MISSING_MEDIUM_VALUE);
            break;
        case greater_than:
            SEARCH_FOR( > , missing_on, MISSING_MEDIUM_VALUE);
            break;
30     case less_than_equal:
            SEARCH_FOR( <= , missing_off, MISSING_MEDIUM_VALUE);
            break;
        case less_than:
            SEARCH_FOR( < , missing_off, MISSING_MEDIUM_VALUE);
            break;
        case not_equal:
            SEARCH_FOR( != , missing_off, MISSING_MEDIUM_VALUE);
            break;
35     case not_equal_list:
            if ( list_head == NULL )
                error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");

            BITMAP_LOOP_PART_1( );
            list_head = compare_list_value;
            found = 0;
            while ( list_head != NULL )
            {

```

```

    if ( list_head->type_list == variable_value )
    {
        if ( *array_start == (unsigned short) list_head->integer )
        {
            found++;
            break;
        }
        if ( *array_start < (unsigned short) list_head->integer )
            break;
    }
    else
    {
        if ( list_head->type_list == range_value )
        {
            if ( (*array_start >= (unsigned short) list_head->between_integer->low) &&
                (*array_start <= (unsigned short) list_head->between_integer->high) )
            {
                found++;
                break;
            }
            if ( *array_start < (unsigned short) list_head->between_integer->low )
                break;
        }
        list_head = list_head->next;
    }
}
/* if value not in the list, select it */
if ( !found )
{
    *temp |= jj;
    total_count++;
}

    array_start++;
    BITMAP_LOOP_PART_2( );
    break;
case between:
    SEARCH_FOR_BETWEEN( );
    break;
default:
    error_handler (INVALID_SWITCH_VALUE, ERROR, operator,
        "operator (default) in search_medium_array_for_value",
        break;
}
return(total_count);
}

/*
** SEARCH_SHORT_ARRAY_FOR_VALUE
** Searches an array of short integer for the given value and updates the
** bitmap
**
**
**
**
*/

int search_short_array_for_value(array_start,compare_value,next_compare_value,
                                compare_list_value,operator,num_bits,results_bitmap)
unsigned char *array_start;
unsigned int compare_value;
unsigned int next_compare_value;
struct list_types *compare_list_value;
enum field_operator operator;
unsigned int num_bits;
struct bitmap *results_bitmap;

```

```

{
    unsigned int total_count = 0;
    unsigned int temp;
    unsigned int i, j, jj, kk;
    unsigned int missing_on=1, missing_off=0;
5   unsigned int missing_flag;
    unsigned int found;

    struct list_types *list_head = NULL;

    list_head = compare_list_value;

    temp = results_bitmap->start;

    switch (operator)
10 {
        case equal:
            SEARCH_FOR( == , missing_off, MISSING_SHORT_VALUE);
            break;
        case equal_list:
            if ( list_head == NULL )
                error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");
            BITMAP_LOOP_PART_1();
15             list_head = compare_list_value;
            while ( list_head != NULL )
            {
                if ( list_head->type_list == variable_value )
                {
                    if ( *array_start == (unsigned char) list_head->integer )
                    {
                        temp |= jj;
                        total_count++;
                        break;
20                     }
                    if ( *array_start < (unsigned char) list_head->integer )
                        break;
                }
                else
                {
                    if ( list_head->type_list == range_value )
                    {
                        if ( (*array_start >= (unsigned char) list_head->between_integer->low) &&
                            (*array_start <= (unsigned char) list_head->between_integer->high) )
25                         {
                            temp |= jj;
                            total_count++;
                            break;
                        }
                        if ( *array_start < (unsigned char) list_head->between_integer->low )
                            break;
                    }
                    list_head = list_head->next;
                }
            }
            array_start++;
30             BITMAP_LOOP_PART_2();
            break;
        case greater_than_equal:
            SEARCH_FOR( >= , missing_on, MISSING_SHORT_VALUE);
            break;
        case greater_than:
            SEARCH_FOR( > , missing_on, MISSING_SHORT_VALUE);
            break;
        case less_than_equal:
35         SEARCH_FOR( <= , missing_off, MISSING_SHORT_VALUE);
            break;
        case less_than:
            SEARCH_FOR( < , missing_off, MISSING_SHORT_VALUE);
            break;
    }
}

```

```

case not_equal:
    SEARCH_FOR( != , missing_off, MISSING_SHORT_VALUE);
    break;
case not_equal_list:
    if ( list_head == NULL )
5      error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");
    BITMAP_LOOP_PART_1( );
    list_head = compare_list_value;
    found = 0;
    while ( list_head != NULL )
    {
        if ( list_head->type_list == variable_value )
        {
            if ( *array_start == (unsigned char) list_head->integer )
10              {
                  found++;
                  break;
              }
            if ( *array_start < (unsigned char) list_head->integer )
                break;
        }
        else
        {
            if ( list_head->type_list == range_value )
15              {
                  if ( (*array_start >= (unsigned char) list_head->between_integer->low) &&
                      (*array_start <= (unsigned char) list_head->between_integer->high) )
                  {
                      found++;
                      break;
                  }
                  if ( *array_start < (unsigned char) list_head->between_integer->low )
                      break;
              }
            list_head = list_head->next;
20          }
        /* if value not in the list, select it */
        if ( !found )
        {
            *temp |= jj;
            total_count++;
        }
25      }

      array_start++;
      BITMAP_LOOP_PART_2( );
      break;
case between:
    SEARCH_FOR_BETWEEN( );
    break;
default:
    error_handler (INVALID_SWITCH_VALUE, ERROR, operator,
30      "operator (default) in search_short_array_for_value");
    break;
    }
    return(total_count);
}

35

```

```

/*
-- SEARCH_BIT_ARRAY_FOR_VALUE
-- Searches an array for bits that have been turned ON or OFF.
5 --
--
--
*/
/* Note: as of July 1993, bit variables are not yet being used and are partially
supported, but none of the bit routines have been tested yet. The list of
parameters for various bit routines may need to be updated. */
int search_bit_array_for_value(array_start,
10 unsigned char *array_start;          operator,num_bits,results_bitmap)
enum field_operator operator;
unsigned int num_bits;
struct bitmap *results_bitmap;
{
    unsigned int total_count = 0;
    unsigned int temp;
    unsigned int i, j, jj, kk;
15    temp = results_bitmap->start;
    switch (operator)
    {
        case equal:
            SEARCH_FOR_BIT_MATCH_EQ( );
            break;
        case not_equal:
            SEARCH_FOR_BIT_MATCH_NEQ( );
            break;
20    case greater_than_equal:
        case greater_than:
        case less_than_equal:
        case less_than:
        case between:
        default:
            error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_bit_array_for_value");
            break;
    }
    return(total_count);
25 }

/*
-- SEARCH_STRING_ARRAY_FOR_VALUE
-- Searches an array of strings for the given value and updates the bitmap
-- Simple case no embedded * or ? wildcard characters, strcmp finds matches
30 -- The latest (and up to date expansion) for all of the macros is found
    in module : FDC_MACRO_DEFN.H
    BITMAP_LOOP_PART_1( ) macro expands like this:
    kk = BITMAP_INTEGER_SIZE;
    jj = 1;
    for ( i=0; i<num_bits; i++)
    {
35 --
    BITMAP_LOOP_PART_2( ) macro expands like this:

```



```

5      if ( --kk )
        if ( --jj )
            else
            {
                temp++;
                jj = 1;
                kk = BITMAP_INTEGER_SIZE;
            }
        }
    }
}

10  /* Notes on string fields relating to MISSING_STRING_VALUE case
    We are choosing to provide no special handling for customer records with
    MISSING_STRING_VALUE data of field length of all space (blank) characters
    Thus a search for NOT_EQUAL "AAA", will get all values NOT_EQUAL to "AAA"
    including " " MISSING. To exclude the missing case you pass, a list as
    in NOT_EQUAL "AAA, ". Please note that this handling for strings works
    differently than the Date, and integer cases, but Mike Emerson has carefully
    considered this and we have chosen this approach. */

15  int search_string_array_for_value(array_start,compare_value,length,
                                     search_value.operator,num_bits,results_bitmap)
    unsigned char *array_start;
    unsigned char *compare_value;
    int length;
    struct list_types *search_value;
    enum field_operator operator;
    unsigned int num_bits;
    struct bitmap *results_bitmap;

20  {
    unsigned int    j=0;
    unsigned int    total_count = 0;
    unsigned int    temp;
    char            *string_value;
    int             string_length;
    unsigned int    i, jj, kk;
    int             found=0;

25  struct list_types *list_head = NULL;

    list_head = search_value;

    /* the strings in the database tables we are looking at are padded with space characters
       and our passed compare_value string is null terminated. Thus we are setting up a
       string of length of database string of spaces and copying the compare_value string
       into the first string_length places so string_value has a space character padded
       version of the compare_value string which we can match successfully the database
       tables. */

30  if ( operator == equal_list || operator == not_equal_list )
    {
        if ( list_head == NULL )
            error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");
        else
            /* length is length of the field not length of the passed match strings */
            length = list_head->string->string_length;

35  /* get all of the list of passed strings to be space padded to proper length */
        /* comparison uses length of entire field, which is space padded in data */
        /* array_start, thus pad passed strings with spaces so compare will work */
        while ( list_head != NULL )

```

```

(
  if ( strlen(list_head->string->string_value) < list_head->string->string_length )
  {
    5      string_length = strlen(list_head->string->string_value);
        string_value = malloc(length+1);
        CHECK_ALLOCATION(string_value, "string_value, Routine search_string_array_for_value()");

        memset( string_value, '\0', length);
        memcpy( string_value, list_head->string->string_value, string_length );
        string_value[length] = '\0';

        free( list_head->string->string_value );
        list_head->string->string_value = string_value;
    10  }
    list_head = list_head->next;

  }
  else
  {
    /* non list case, we have compare_value string and length passed */
    string_length = strlen(compare_value);

    15  /* intention is to pad the string with blanks, upto the "length" of the string */
        string_value = malloc(length+1);
        CHECK_ALLOCATION(string_value, "string_value, Routine search_string_array_for_value()");

        if ( string_length < length )
        {
            memset( string_value, '\0', length );
            memcpy( string_value, compare_value, string_length );
        }
        else
            /* copying only part of compare_value string */
    20      memcpy( string_value, compare_value, length );

        string_value[length] = '\0';
    }

    temp = results_bitmap->start;

    switch (operator)
    {
    25      case equal
        BITMAP_LOOP_PART_1( );
        if (strcmp(string_value, array_start, length) == 0)
        {
            *temp |= jj;
            total_count++;
        }
        array_start += length;
        BITMAP_LOOP_PART_2( );
        break;
    30      case equal_list:
        BITMAP_LOOP_PART_1( );
        list_head = search_value;
        while ( list_head != NULL )
        {
            if ( (strcmp( list_head->string->string_value, array_start, length)) == 0 )
            {
                *temp |= jj;
                total_count++;
                break;
            }
            35      list_head = list_head->next;
        }
    }
  }
}

```

```

        array_start += length;
        BITMAP_LOOP_PART_2( );
        break;
    case not_equal:
        BITMAP_LOOP_PART_1( );
        if (strcmp(string_value, array_start, length) != 0)
        {
            temp |= jj;
            total_count++;
        }
        array_start += length;
        BITMAP_LOOP_PART_2( );
        break;
    case not_equal_list:
        BITMAP_LOOP_PART_1( );
        found = 0;
        list_head = search_value;
        while (list_head != NULL)
        {
            if (strcmp(list_head->string->string_value, array_start, length) == 0)
            {
                found++;
                break;
            }
            list_head = list_head->next;
        }
        if (found)
        {
            temp |= jj;
            total_count++;
        }
        array_start += length;
        BITMAP_LOOP_PART_2( );
        break;
    default:
        error_handler(INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_string_array_for_value");
        break;
}

if (operator != equal_list && operator != not_equal_list)
{
    free(string_value);
    string_value = NULL;
}

return(total_count);
}

/* fixed string search similar to above but uses fixed string data
constructs. This is the simple match case, compare value has
no * or ? characters
*/

int search_fstring_array_for_value(array_start, compare_str, length, field_offset,
                                search_value, operator, num_bits, results_bitmap)
unsigned short *array_start;
unsigned char *compare_str;
int length;
int field_offset;
struct list_types *search_value;
enum field_operator operator;
unsigned int num_bits;
struct bitmap *results_bitmap;
{

```

```

    unsigned int j=0;
    unsigned int total_count = 0;
    unsigned int temp;
    unsigned int compare_value = MISSING_MEDIUM_VALUE;
    unsigned int i, j, kk;
5   int          indx=0;
    unsigned int missing_on = 1, missing_off = 0;
    unsigned int missing_flag=missing_off;
    int          found=0;

    struct list_types *list_head = NULL;

    list_head = search_value;

10  if ( operator != equal_list && operator != not_equal_list )
    {
        /* only 1 item to be found, string in compare_str */
        for (indx=1; indx<=fixed_field(field_offset)->indx_count; indx++)
        {
            if (fixed_field(field_offset)->fixed_string[indx] != NULL)
            {
                if (strcmp(fixed_field(field_offset)->fixed_string[indx]->string, compare_str) == 0)
                {
                    compare_value = indx;
                    break;
25  }
                }
            }
        }
        /* deal with missing. MISSING_STRING_VALUE text string is not in fixed_field tables */
        /* but data array has MISSING_MEDIUM_VALUE for all missing fixed strings */
        if (strcmp(compare_str, MISSING_STRING_VALUE, strlen(MISSING_STRING_VALUE)) == 0)
            compare_value = MISSING_MEDIUM_VALUE;
    }
    /* deal with chain list of strings without wild cards */
20  else
    {
        /* list of strings, in chained list_types list, translate each from string to indx, */
        /* put indx into list_types structures */
        if ( list_head == NULL )
            error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");
        else
            length = list_head->string->string_length;

25  /* first deal with missing_string_value in chained list, uses indx of MISSING_MEDIUM_VALUE */
        while ( list_head != NULL )
        {
            list_head->integer = 0;
            if ( strcmp( list_head->string->string_value, MISSING_STRING_VALUE,
                strlen(MISSING_STRING_VALUE)) == 0)

30  {
                list_head->integer = MISSING_MEDIUM_VALUE;
                compare_value = MISSING_MEDIUM_VALUE;          /* so can tell found at least one */
            }
            list_head = list_head->next;
        }

        /* now match strings in chained list with strings in fixed_strings structure, keeping indx */
        /* values which will match items in data array */
        for (indx=1; indx<=fixed_field(field_offset)->indx_count; indx++)
35  {
            if (fixed_field(field_offset)->fixed_string[indx] != NULL)
            {
                list_head = search_value;
                while ( list_head != NULL )
            {

```

```

        if (strcmp(fixed_field[field_offset] -> fixed_string(indx) -> string, list_head -> string -> string_value) == 0)
        {
            compare_value = indx;
            list_head -> integer = indx;
            break;
        }
        list_head = list_head -> next;
    }
}

temp = results_bitmap -> start;

/* if missing we found no matching strings in table of fixed string text, no need to search */
/* except for not equal cases */
10 if (compare_value != MISSING_MEDIUM_VALUE || operator == not_equal
    || operator == not_equal_list)
{
    switch (operator)
    {
        case equal:
            SEARCH_FOR( ==, missing_off, MISSING_MEDIUM_VALUE);
            break;
        case not_equal:
15 SEARCH_FOR( !=, missing_off, MISSING_MEDIUM_VALUE);
            break;
        case equal_list:
            BITMAP_LOOP_PART_1();
            list_head = search_value;
            while (list_head != NULL)
            {
                if (*array_start == (unsigned short) list_head -> integer)
                {
20 temp |= y;
                    total_count++;
                    break;
                }
                list_head = list_head -> next;
            }
            array_start++;
            BITMAP_LOOP_PART_2();
            break;
        case not_equal_list:
25 BITMAP_LOOP_PART_1();
            found = 0;
            list_head = search_value;
            while (list_head != NULL)
            {
                if (*array_start == (unsigned short) list_head -> integer)
                {
30 found++;
                    break;
                }
                list_head = list_head -> next;
            }
            /* exclude MISSING_STRING_VALUE items (val is MISSING_MEDIUM_VALUE) */
            /* confusing but correct, if MISSING_MEDIUM_VALUE is in the list */
            /* will find and not put in the list, thus is MISSING, & not found */
            /* must exclude it here */
            if (!found && *array_start != MISSING_MEDIUM_VALUE)
            {
35 temp |= y;
                total_count++;
            }
            array_start++;
            BITMAP_LOOP_PART_2();
            break;
    }
}

```

```

/*
 * SEARCH_MIXED_STRING_ARRAY_FOR_VALUE
 *
 * Searches an array of strings for the given value and updates the bitmap
 * The string or at least one of the series to strings with values to look
 * for have embedded * and/or ? wild card characters.
 */
int search_mixed_string_for_value(array_start, compare_value, length,
                                search_value, operator, num_bits, results_bitmap)
    unsigned char *array_start;
    unsigned char *compare_value;
    int length;
    struct list_types *search_value;
    enum field_operator operator;
    unsigned int num_bits;
    struct bitmap *results_bitmap;
    {
        unsigned int      j=0;
        unsigned int      total_count = 0;
        unsigned int      *temp;
        unsigned int      i, j, kk;
        int               found=0;
        int               missing_flag = 0;

        struct list_types *list_head = NULL;

        list_head = search_value;

        if ( operator == equal_list || operator == not_equal_list )
        {
            if ( list_head == NULL )
            20 error_handler(NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");
            else
                /* length is length of field, not just the passed string(s) */
                length = list_head->string->string_length;

            #ifdef no_special_string_searches_for_MISSING_STRING_VALUE
                /* see if list contains entire MISSING_STRING_VALUE entry */
                while ( list_head != NULL )
                {
                    25 if( strcmp(list_head->string->string_value, MISSING_STRING_VALUE,
                                strlen(MISSING_STRING_VALUE)) == 0)
                    {
                        /* mark MISSING_STRING_VALUE as in the list */
                        missing_flag++;
                        break;
                    }
                    list_head = list_head->next;
                }
            30 #endif /* no_special_string_searches_for_MISSING_STRING_VALUE */

            temp = results_bitmap->start;

            switch (operator)
            {
                case equal:
                    35 BITMAP_LOOP_PART_1( );
                        if ( (compare_string(array_start, compare_value, length)) )
                        {
                            *temp |= j;
                            total_count++;
                        }
                    }
            }
        }
    }

```

```

    )
    array_start += length;
    BITMAP_LOOP_PART_2( );
    break;
5 case equal_list:
    BITMAP_LOOP_PART_1( );
    list_head = search_value;
    while ( list_head != NULL )
    {
        if ( compare_string(array_start, list_head->string->string_value, length) )
        {
            temp |= jj;
            total_count++;
            break;
10        }
        list_head = list_head->next;
    }
    array_start += length;
    BITMAP_LOOP_PART_2( );
    break;
case not_equal:
    BITMAP_LOOP_PART_1( );
    if (!(compare_string(array_start, compare_value, length)))
15    {
        temp |= jj;
        total_count++;
    }
    array_start += length;
    BITMAP_LOOP_PART_2( );
    break;
case not_equal_list:
    BITMAP_LOOP_PART_1( );
    found = 0;
20    list_head = search_value;
    while ( list_head != NULL )
    {
        if ( compare_string(array_start, list_head->string->string_value, length) )
        {
            found++;
            break;
        }
        list_head = list_head->next;
25    }
    if ( !found )
    {
        temp |= jj;
        total_count++;
    }
    array_start += length;
    BITMAP_LOOP_PART_2( );
    break;
30 case greater_than_equal:
case greater_than:
case less_than_equal:
case less_than:
case between:
default:
    error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_mixed_string_for_value");
    break;
}
35 return(total_count);
}

```

```

int search_mixed_string_for_value(array_start,compare_str,length,field_offset,
                                search_value,operator,num_bits,results_bitmap)
unsigned short *array_start;
unsigned char *compare_str;
int length;
5 int field_offset;
struct list_types *search_value;
enum field_operator operator;
int num_bits;
struct bitmap *results_bitmap;
{
    unsigned int j=0;
    unsigned int total_count = 0;
    unsigned int temp;
10 unsigned int compare_value = MISSING_MEDIUM_VALUE;
    unsigned int i, j, kk;
    int indx;
    int found=0;
    int missing_flag = 0;

    struct list_types *list_head = NULL;

    list_head = search_value;

15 temp = results_bitmap->start;

    switch (operator)
    {
        case equal:
            for (indx=1; indx<=fixed_field(field_offset)->indx_count; indx++)
            {
                if (fixed_field(field_offset)->fixed_string[indx] != NULL)
20
                /* note: wild card case. process entire list of fixed_fields, finding all matches */
                if ( compare_string(fixed_field(field_offset)->fixed_string[indx]->string, compare_str, length) )
                {
                    fixed_field(field_offset)->fixed_string[indx]->bit_select++;
                    compare_value = 1;
                }
            }
25 if ( compare_value != MISSING_MEDIUM_VALUE )
        {
            BITMAP_LOOP_PART_1( ).
            /* fixed_field(field_offset)->fixed_string is an array of FIXED_STRING_MAX_INDEX, all pointing */
            /* to NULL, except for the values that are being used. So, the missing value of fixed_string */
            /* will point to NULL */
            /* mixed string equal case will not handle MISSING as this value must be fully named, not wild */
            /* carded, and thus wont be in mixed fixed string one item match case */
30 if (fixed_field(field_offset)->fixed_string[*array_start] != NULL)
            {
                if (fixed_field(field_offset)->fixed_string[*array_start]->bit_select)
                {
                    temp |= j;
                    total_count++;
                }
            }
            array_start++;
35 BITMAP_LOOP_PART_2( );
        }
    }
    break;

```



```

case equal_list:
    if ( list_head == NULL )
        error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");
        /* see if looking for missing values
        /* list item must have string matching symbol MISSING_STRING_VALUE spelled out */
        /* entirely, not allowing any wild card matches for missing */
        missing_flag = 0;
5      while ( list_head != NULL )
        {
            if ( strcmp( list_head->string->string_value, MISSING_STRING_VALUE, strlen(MISSING_STRING_VALUE)) == 0 )
            {
                missing_flag++;
                break;
            }
            list_head = list_head->next;
        }
10     for (indx=1; indx<=fixed_field[field_offset]->indx_count; indx++)
        {
            if (fixed_field[field_offset]->fixed_string[indx] != NULL)
            {
                list_head = search_value;
                while ( list_head != NULL )
                {
                    if ( compare_string(fixed_field[field_offset]->fixed_string[indx]->string,
15                     list_head->string->string_value, list_head->string->string_length) )
                    {
                        fixed_field[field_offset]->fixed_string[indx]->bit_select++;
                        compare_value = 1; /* note found at least one, so search is valid */
                        break;
                    }
                    list_head = list_head->next;
                }
            }
        }
        if ( compare_value != MISSING_MEDIUM_VALUE )
20     {
        BITMAP_LOOP_PART_1( );
        if (fixed_field[field_offset]->fixed_string[*array_start] != NULL)
        {
            if (fixed_field[field_offset]->fixed_string[*array_start]->bit_select)
            {
                *temp |= j;
                total_count++;
                continue;
            }
        }
25     }
        /* also check if to pick up missing values also */
        if ( missing_flag )
        {
            if ( *array_start == MISSING_MEDIUM_VALUE )
            {
                *temp |= j;
                total_count++;
            }
        }
        *array_start++;
        BITMAP_LOOP_PART_2( );
30     }
        break;
case not_equal:
    for (indx=1; indx<=fixed_field[field_offset]->indx_count; indx++)
    {
        if (fixed_field[field_offset]->fixed_string[indx] != NULL)
        {
            if ( !compare_string(fixed_field[field_offset]->fixed_string[indx]->string, compare_str, length) )
            {
35             fixed_field[field_offset]->fixed_string[indx]->bit_select++;
            }
        }
    }

```

```

    }
    if ( compare_value != MISSING_MEDIUM_VALUE )
    {
        BITMAP_LOOP_PART_1( );
        if (fixed_field[field_offset]->fixed_string[*array_start] != NULL)
        {
            if (fixed_field[field_offset]->fixed_string[*array_start]->bit_select)
            {
                temp += 1;
                total_count++;
            }
        }
        array_start++;
        BITMAP_LOOP_PART_2( );
    }
    break;
case not_equal_list:
    if ( list_head == NULL )
        error_handler (NULL_POINTER_ERR, ERROR, NO_STATUS, "list_head in search_section");

    /* don't have to deal with missing here as we won't include any missing values whether
    /* MISSING is one of the passed strings or not.

    for (indx=1; indx<=fixed_field[field_offset]->indx_count; indx++)
    {
        found = 0;
        if (fixed_field[field_offset]->fixed_string[indx] != NULL)
        {
            list_head = search_value;
            while ( list_head != NULL )
            {
                if ( compare_string(fixed_field[field_offset]->fixed_string[indx]->string
                    list_head->string->string_value, list_head->string->string_length) )
                {
                    found++;
                    break;
                }
                list_head = list_head->next;
            }
            if ( !found )
            {
                fixed_field[field_offset]->fixed_string[indx]->bit_select++;
                compare_value = 1;
            }
        }
    }
    if ( compare_value != MISSING_MEDIUM_VALUE )
    {
        BITMAP_LOOP_PART_1( )
        if (fixed_field[field_offset]->fixed_string[*array_start] != NULL)
        {
            /* we have marked bit_select for items not matching the list */
            if (fixed_field[field_offset]->fixed_string[*array_start]->bit_select)
            {
                temp += 1;
                total_count++;
            }
        }
        array_start++;
        BITMAP_LOOP_PART_2( );
    }
    break;
default:
    error_handler (INVALID_SWITCH_VALUE, ERROR, operator,
        "operator (default) in search_mixed_tstring_for_value");
    break;
}

```

```

/* initialize the status flag */
if ( compare_value != MISSING_MEDIUM_VALUE )
{
    for (indx=1; indx<=fixed_field(field_offset)->indx_count; indx++)
    {
        /* must verify an allocated data structure */
        if (fixed_field(field_offset)->fixed_string[indx] != NULL)
            fixed_field(field_offset)->fixed_string[indx]->bit_select = 0;
    }
    return(total_count);
}

/*
10 SEARCH_LARGE_FIELD_ARRAY
-- Used, when the values of two fields are compared.
-- For a given value in array 1, it searches all values in array 2 for the
-- occurrence.
-- SEARCH_PROCESS_ARRAY expands as:
15 if ( *array_start oper *array_start1 )
{
    temp |= jj;
    temp1 |= jj;
    total_count++;
}
array_start++;
array_start1++;
-- with oper being substituted for with ==, >=, !=, etc
20 */

int search_large_field_array(array_start, array_start1, operator,
                           num_bits, results_bitmap)
unsigned int *array_start;
unsigned int *array_start1;
enum field_operator operator;
int num_bits;
struct bitmap *results_bitmap;
25 {
    unsigned int j;
    unsigned int total_count = 0;
    unsigned int temp;
    unsigned int i, jj, kk;

    temp = results_bitmap->start;

    switch (operator)
    30 {
        case equal
            BITMAP_LOOP_PART_1();
            if ( *array_start == *array_start1 && *array_start < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;

        case greater_than
            BITMAP_LOOP_PART_1();
            if ( *array_start > *array_start1 && *array_start < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;
    }
    35
}

```

```

    case greater_than_equal:
        BITMAP_LOOP_PART_1();
        if ( *array_start >= *array_start1 && *array_start < MISSING_LARGE_VALUE )
            SEARCH_PROCESS_ARRAY();
        BITMAP_LOOP_PART_2();
        break;

    case less_than_equal:
        BITMAP_LOOP_PART_1();
        if ( *array_start <= *array_start1 && *array_start1 < MISSING_LARGE_VALUE )
            SEARCH_PROCESS_ARRAY();
        BITMAP_LOOP_PART_2();
        break;

    case less_than:
        BITMAP_LOOP_PART_1();
        if ( *array_start < *array_start1 && *array_start1 < MISSING_LARGE_VALUE )
            SEARCH_PROCESS_ARRAY();
        BITMAP_LOOP_PART_2();
        break;

    case not_equal:
        BITMAP_LOOP_PART_1();
        if ( *array_start != *array_start1 &&
            *array_start < MISSING_LARGE_VALUE &&
            *array_start1 < MISSING_LARGE_VALUE )
            SEARCH_PROCESS_ARRAY();
        BITMAP_LOOP_PART_2();
        break;

    case between:
    default:
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_large_field_array");
        break;

    }
    return(total_count);
}

/*
-- SEARCH_MEDIUM_FIELD_ARRAY
--
-- Used, when the values of two fields are compared.
-- For a given value in array 1, it searches all values in array 2 for the
-- occurrence.
--
--
--
--
*/
int search_medium_field_array(array_start, array_start1, operator,
                             num_bits, results_bitmap)
30 ( unsigned short *array_start;
    unsigned short *array_start1;
    enum field_operator operator;
    int num_bits;
    struct bitmap *results_bitmap;
    {
        unsigned int j;
        unsigned int total_count = 0;
        unsigned int *temp;
        35 unsigned int i, jj, kk;

        temp = results_bitmap->start;

        switch (operator)
        {

```

```

case equal:
    BITMAP_LOOP_PART_1();
    if ( *array_start == *array_start1 && *array_start < MISSING_MEDIUM_VALUE )
        SEARCH_PROCESS_ARRAY();
    BITMAP_LOOP_PART_2();
    break;
case greater_than_equal:
5   BITMAP_LOOP_PART_1();
    if ( *array_start >= *array_start1 && *array_start < MISSING_MEDIUM_VALUE )
        SEARCH_PROCESS_ARRAY();
    BITMAP_LOOP_PART_2();
    break;
case greater_than:
    BITMAP_LOOP_PART_1();
    if ( *array_start > *array_start1 && *array_start < MISSING_MEDIUM_VALUE )
        SEARCH_PROCESS_ARRAY();
10   BITMAP_LOOP_PART_2();
    break;
case less_than_equal:
    BITMAP_LOOP_PART_1();
    if ( *array_start <= *array_start1 && *array_start1 < MISSING_MEDIUM_VALUE )
        SEARCH_PROCESS_ARRAY();
    BITMAP_LOOP_PART_2();
    break;
case less_than:
15   BITMAP_LOOP_PART_1();
    if ( *array_start == *array_start1 && *array_start1 < MISSING_MEDIUM_VALUE )
        SEARCH_PROCESS_ARRAY();
    BITMAP_LOOP_PART_2();
    break;
case not_equal:
    BITMAP_LOOP_PART_1();
    if ( *array_start != *array_start1 && *array_start < MISSING_MEDIUM_VALUE && *array_start1
        < MISSING_MEDIUM_VALUE )
        SEARCH_PROCESS_ARRAY();
20   BITMAP_LOOP_PART_2();
    break;
case between:
default
    error_handler( INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_medium_field_array",
        break;
    )
return( total_count );
}

25  /*
    -- SEARCH_SHORT_FIELD_ARRAY
    -- Used, when the values of two fields are compared.
    -- For a given value in array 1, it searches all values in array 2 for the
    -- occurrence.
    --
    --
    --
    --
    */
30  int search_short_field_array( array_start, array_start1, operator,
                                num_bits, results_bitmap )
    unsigned char *array_start;
    unsigned char *array_start1;
    enum field_operator operator;
    int num_bits;
    struct bitmap *results_bitmap;
    {
35     unsigned int j;
        unsigned int total_count = 0;
        unsigned int temp;
        unsigned int i, j, kk;

        temp = results_bitmap->start;

        switch ( operator )
        {

```

```

case equal:
    BITMAP_LOOP_PART_1( );
    if ( *array_start == *array_start1 && *array_start < MISSING_SHORT_VALUE )
        SEARCH_PROCESS_ARRAY( );
    BITMAP_LOOP_PART_2( );
    break;
5 case greater_than_equal:
    BITMAP_LOOP_PART_1( );
    if ( *array_start >= *array_start1 && *array_start < MISSING_SHORT_VALUE )
        SEARCH_PROCESS_ARRAY( );
    BITMAP_LOOP_PART_2( );
    break;
case greater_than:
    BITMAP_LOOP_PART_1( );
    if ( *array_start > *array_start1 && *array_start < MISSING_SHORT_VALUE )
10 SEARCH_PROCESS_ARRAY( );
    BITMAP_LOOP_PART_2( );
    break;
case less_than_equal:
    BITMAP_LOOP_PART_1( );
    if ( *array_start <= *array_start1 && *array_start1 < MISSING_SHORT_VALUE )
        SEARCH_PROCESS_ARRAY( );
    BITMAP_LOOP_PART_2( );
    break;
15 case less_than:
    BITMAP_LOOP_PART_1( );
    if ( *array_start < *array_start1 && *array_start1 < MISSING_SHORT_VALUE )
        SEARCH_PROCESS_ARRAY( );
    BITMAP_LOOP_PART_2( );
    break;
case not_equal:
    BITMAP_LOOP_PART_1( );
    if ( *array_start != *array_start1 && *array_start < MISSING_SHORT_VALUE
20 && *array_start1 < MISSING_SHORT_VALUE )
        SEARCH_PROCESS_ARRAY( );
    BITMAP_LOOP_PART_2( );
    break;

case between:
default:
    error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_short_field_array");

25 break;
    }
    return(total_count);
}

/*
-- SEARCH_DOUBLE_FIELD_ARRAY
30 -- Used, when the values of two fields are compared.
-- For a given value in array 1, it searches all values in array 2 for the
-- occurrence.
--
--
--
*/

35 /* Note: as of May 1993, doubles are not yet actually supported. To use this we will need a
MISSING_DOUBLE_VALUE to be defined and put into the if statements in place of MISSING_LARGE_VALUE */
int search_double_field_array(array_start, array_start1, operator,
                             num_bns, results_bitmap)

```

```

double *array_start;
double *array_start1;
enum field_operator operator;
int num_bits;
struct bitmap *results_bitmap;
5 {
    unsigned int j;
    unsigned int total_count = 0;
    unsigned int *temp;
    unsigned int *temp1;
    unsigned int i, j, kx;

    temp = results_bitmap->start;

10    switch (operator)
    {
        case equal:
            BITMAP_LOOP_PART_1();
            if ( *array_start == *array_start1 && *array_start < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;

        case greater_than_equal:
15            BITMAP_LOOP_PART_1();
            if ( *array_start >= *array_start1 && *array_start < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;

        case greater_than:
            BITMAP_LOOP_PART_1();
            if ( *array_start > *array_start1 && *array_start < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;

20        case less_than_equal:
            BITMAP_LOOP_PART_1();
            if ( *array_start <= *array_start1 && *array_start1 < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;

        case less_than:
25            BITMAP_LOOP_PART_1();
            if ( *array_start < *array_start1 && *array_start1 < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;

        case not_equal:
            BITMAP_LOOP_PART_1();
            if ( *array_start != *array_start1 && *array_start
                < MISSING_LARGE_VALUE && *array_start1 < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
30            break;

        case between:
        default:
            error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_double_field_array");

            break;

35    }
    return(total_count);
}

```

```

/*
-- SEARCH_FLOAT_FIELD_ARRAY
-- Used, when the values of two fields are compared.
-- For a given value in array 1, it searches all values in array 2 for the
5 -- occurrence.
--
--
*/

/* Note: as of May 1993, floats are not yet actually supported. To use this we will need a
MISSING_FLOAT_VALUE to be defined and put into the if statements in place of MISSING_LARGE_VALUE */

10 int search_float_field_array(array_start, array_start1, operator,
                                num_bits, results_bitmap)
    float *array_start;
    float *array_start1;
    enum field_operator operator;
    int num_bits;
    struct bitmap *results_bitmap;
{
    unsigned int j;
    unsigned int total_count = 0;
15   unsigned int temp;
    unsigned int i, jj, kk;

    temp = results_bitmap->start;

    switch (operator)
    {
        case equal:
20         BITMAP_LOOP_PART_1();
            if ( *array_start == *array_start1 && *array_start < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;
        case greater_than_equal:
            BITMAP_LOOP_PART_1();
            if ( *array_start >= *array_start1 && *array_start < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;
25         case greater_than:
            BITMAP_LOOP_PART_1();
            if ( *array_start > *array_start1 && *array_start < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;
        case less_than_equal:
30         BITMAP_LOOP_PART_1();
            if ( *array_start <= *array_start1 && *array_start1 < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;
        case less_than:
            BITMAP_LOOP_PART_1();
            if ( *array_start < *array_start1 && *array_start1 < MISSING_LARGE_VALUE )
                SEARCH_PROCESS_ARRAY();
            BITMAP_LOOP_PART_2();
            break;
35
    }
}

```



```

    case not_equal:

5  BITMAP_LOOP_PART_1( ):
    if ( *array_start != *array_start1 && *array_start < MISSING_LARGE_VALUE && *array_start1 < MISSING_LARGE_VALUE )
        SEARCH_PROCESS_ARRAY( );
    BITMAP_LOOP_PART_2( );
    break;

    case between:
    default:
        error_handler( INVALID_SWITCH_VALUE, ERROR, operator, "operator in search float_field_array" );

10         break;

    )
    return( total_count );

)

15 /*
   SEARCH_BIT_FIELD_ARRAY
   -- Used, when the values of two fields are compared
   -- For a given value in array 1, it searches all values in array 2 for the
   -- occurrence.
   --
   --
   --
20 */

/* Note, as of May 1993, bits are not yet actually supported. */

int search_bit_field_array( array_start, array_start1, operator,
                           num_bits, results_bitmap )
    unsigned char *array_start,
    unsigned char *array_start1,
    enum field_operator operator,
    int num_bits,
25 struct bitmap *results_bitmap;

/* this needs to be rewritten dependent on the new load of two bits for bits */

{
    unsigned int j;
    unsigned int total_count = 0;
    unsigned int temp;
    unsigned int i, jj, kk;
30 temp = results_bitmap->start;

    switch ( operator )
    {
        case equal:
            BITMAP_LOOP_PART_1( );
            if ( (*array_start & jj) && (*array_start1 & jj) )
            {
                temp |= jj;
                total_count++;
35             }
    }
}

```

```

        array_start++;
        array_start1++;
        BITMAP_LOOP_PART_2( );
        break;
5      case not_equal:
        BITMAP_LOOP_PART_1( );
        if (!(*array_start & j) && (*array_start1 & j))
        {
            temp |= j;
            total_count++;
        }
        array_start++;
        array_start1++;
        BITMAP_LOOP_PART_2( );
10      break;
      case greater_than_equal:
      case greater_than:
      case less_than_equal:
      case less_than:
      case between:
        /* Note for these cases, possibly we will want different failure action than default case */
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "invalid operator in search_bit_field_array");
        break;
15      default:
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (default) in search_bit_field_array");
        break;
    }
    return(total_count);
}

20  /*
    -- SEARCH_STRING_FIELD_ARRAY
    -- Used, when the values of two fields are compared
    -- For a given value in array 1, it searches all values in array 2 for the
    -- occurrence
    --
    --
    --
25  */

int search_string_field_array(          array_start,
                                     array_start1,
                                     operator,
                                     num_bits,
                                     length,
                                     results_bitmap)

    unsigned char *array_start,
    unsigned char *array_start1,
30  enum field_operator operator,
    int num_bits,
    int length,
    struct bitmap *results_bitmap;

{
    unsigned int      j;
    unsigned int      total_count = 0;
    unsigned int      temp;
35  unsigned int      i, j, k;

```

```

temp = results_bitmap->start;
switch (operator)
{
    case equal:
        BITMAP_LOOP_PART_1( );
        if (strcmp(array_start, array_start1, length)==0)
        {
            temp |= jj;
            total_count++;
        }
        array_start += length;
        array_start1 += length;
        BITMAP_LOOP_PART_2( );
        break;
    case not_equal:
        BITMAP_LOOP_PART_1( );
        if (strcmp(array_start, array_start1, length)!=0)
        {
            temp |= jj;
            total_count++;
        }
        array_start += length;
        array_start1 += length;
        BITMAP_LOOP_PART_2( );
        break;
    case greater_than_equal:
    case greater_than:
    case less_than_equal:
    case less_than:
    case between:
        /* in case we want different future fail action for these operators */
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "invalid operator in search_string_field_array")
        break;
    default:
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (default) in search_string_field_array");
        break;
}
return(total_count)
}

/*
 * SEARCH_FSTRING_FIELD_ARRAY
 */

int search_fstring_field_array(array_start, array_start1, operator, num_bits, length,
                               f_offset, f_offset1, results_bitmap)

unsigned short *array_start;
unsigned short *array_start1;
enum field_operator operator;
int num_bits;
int length;
int f_offset;
int f_offset1;
struct bitmap *results_bitmap;
{
    unsigned int      j;
    unsigned int      total_count = 0;
    unsigned int      temp, res;
    unsigned int      i, jj, kk;

```

```

temp = results_bitmap->start;
res = results_bitmap->start;
5 switch (operator)
{
    case equal:
        BITMAP_LOOP_PART_1();
        /* string_n_compare looks at string that is found using array_start.1 as
        indices, using fixed_field table with passed f_offset index */

        /*
        10 syp - dan, this should be done, since the values in the database should map
        to what is in our list?
        (Note: if *array_start or *array_start1 point to a null structure, query engine
        crashes with an access violation. step likely unneeded but provides some safety
        */
        if ((fixed_field[f_offset]->fixed_string[*array_start] != 0) &&
            (fixed_field[f_offset1]->fixed_string[*array_start1] != 0))
        {
            15 if ((strcmp(fixed_field[f_offset]->fixed_string[*array_start]->string,
            fixed_field[f_offset1]->fixed_string[*array_start1]->string.length)) == 0)
            {
                temp |= 1;
                total_count++;
            }
        }
        array_start++;
        array_start1++;
        BITMAP_LOOP_PART_2();
        break;

    20 case not_equal:
        BITMAP_LOOP_PART_1();
        /* string_n_compare looks at string that is found using array_start.1 as
        indices, using fixed_field table with passed f_offset index */

        if ((fixed_field[f_offset]->fixed_string[*array_start] != 0) &&
            (fixed_field[f_offset1]->fixed_string[*array_start1] != 0))
        {
            25 if ((strcmp(fixed_field[f_offset]->fixed_string[*array_start]->string,
            fixed_field[f_offset1]->fixed_string[*array_start1]->string.length)) != 0)
            {
                temp |= 1;
                total_count++;
            }
        }
        array_start++;
        array_start1++;
        BITMAP_LOOP_PART_2();
        break;

    30 case greater_than_equal:
    case greater_than:
    case less_than_equal:
    case less_than:
    case between:
        /* in case we want different future fail action for these operators */
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "invalid operator in search_fstring_field_array");
        break;

    default:
        35 error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (default) in search_fstring_field_array");
        break;
}
return(total_count);
}

```

```

/*
 * CHECK_LIST_FOR_MIXED_STRINGS
 *
 * checks a passed list of chained list_types data structures with character
 * strings pointers for any * or ? wild card mixed string indicators.
 * if any * or ? char is found, returns 1 for mixed strings. if end of list
5  * is found without wild cards, returns 0.
 *
 * Used for both fixed_string and string cases. Both are passed list with
 * character strings to match.
 */

int check_list_for_mixed_strings( search_list )
struct list_types *search_list;
10 {
    struct list_types *list_head;

    list_head = search_list;

    while ( list_head != NULL )
    {
        /* return 1 if wild card is found */
        if ( (strchr(list_head->string->string_value, '?') != NULL) ||
15         (strchr(list_head->string->string_value, '*') != NULL) )
            return( 1 );

        list_head = list_head->next;
    }

    /* no mixed string wild card characters found, return plain string code 0 */
    return(0);
}

20 /*
 * SEARCH_DATASET_FOR_VALUE
 *
 * Calls the appropriate routines to build the results based on their field
 * types.
 *
 *
 *
25 */

int search_dataset_for_value( data

                                data1,
                                field_type,
                                field_size,
                                field_offset,
                                field_offset1,
                                operator,
                                search_value,
                                search_value_type,
                                results_bitmap)

30 struct dataset *data,
    struct dataset *data1,
    enum data_type field_type,
    int field_size,
    int field_offset,
    int field_offset1,
    enum field_operator operator,
    union search_item search_value,
35 enum search_value_item search_value_type,
    struct bitmap *results_bitmap;

{

```

```

    unsigned int total_count = 0;
    int num_bits;

5   num_bits = data->number_of_items;

    switch (field_type)
    {
        case dollars:
        case floating_point:
        case large_integer:
            if (search_value_type == field_value)
            {
10              total_count = search_large_field_array(

                                data->items
                                data1->items,
                                operator,
                                num_bits,
                                results_bitmap);
            }
        else
        {
15          if (search_value_type == list_value)
            /* we are looking at a list of values */
            {
                total_count = search_large_array_for_value(

                                data->items
                                (unsigned int)0
                                (unsigned int)0
                                search_value list
                                operator,
                                data->number_of_items
                                results_bitmap);
            }
        }
20      else
        {
            if (operator == between)
                total_count = search_medium_array_for_value(

                                data->items,
                                search_value.between_integer->low,
                                search_value.between_integer->high,
                                NULL,
                                operator,
                                data->number_of_items,
                                results_bitmap);
            else
25              total_count = search_large_array_for_value(

                                data->items
                                search_value integer,
                                (unsigned int)0,
                                NULL,
                                operator,
                                data->number_of_items,
                                results_bitmap);
        }
    }
    break;
    case year_month:
    case year_month_day:
    case medium_integer:
30      if (search_value_type == field_value)
      {
          total_count = search_medium_field_array(

                                data->items,
                                data1->items,
                                operator,

```

```

    }
    else
    {
        if (search_value_type == list_value)
        /* we are looking at a list of values */
5      {
          total_count = search_medium_array_for_value(

          data->items,
          (unsigned int)0,
          (unsigned int)0,
          search_value.list,
          operator,
          data->number_of_items,
          results_bitmap);

10      }
      else
      {
          if (operator==between)
              total_count = search_medium_array_for_value(

          data->items,
          search_value.between_integer->low,
          search_value.between_integer->high,
          NULL,
          operator,
          data->number_of_items,
          results_bitmap);

15      else
          total_count = search_medium_array_for_value(

          data->items,
          search_value.integer,
          (unsigned int)0,
          NULL,
          operator,
          data->number_of_items,
          results_bitmap);

20      }
    }
    break;
case character
case small_integer
    if (search_value_type == field_value)
    {
        total_count = search_short_field_array(

25      data->items
        data1->items,
        operator,
        num_bits
        results_bitmap)
    }
    else
    {
        if (search_value_type == list_value)
        /* we are looking at a list of values */
30      {
          total_count = search_short_array_for_value(

          data->items
          (unsigned int)0,
          (unsigned int)0,
          search_value.list,
          operator,
          data->number_of_items,
          results_bitmap);

35      }
    }
    else
    {

```

```

5      if (operator == between)
        total_count = search_short_array_for_value(
            data->items,
            search_value between_integer->low,
            search_value between_integer->high,
            NULL,
            operator,
            data->number_of_items,
            results_bitmap);

        else
            total_count = search_short_array_for_value(
                data->items,
                search_value integer,
                (unsigned int)0,
                NULL,
                operator,
                data->number_of_items,
                results_bitmap);

        }
        break;

    case fixed_string
        if (search_value_type == field_value)
        {
15            total_count = search_fstring_field_array(
                data->items,
                data1->items,
                operator,

                num_bits,
                field_size,
                field_offset,
                field_offset1,

                results_bitmap);

20        }
        else
        {
            if (search_value_type == list_value)
            /* we are looking at a list of values */
            {
                /* see if any mixed strings or not */
                if ( check_list_for_mixed_strings( search_value list ) )
                {
25                    /* mixed strings case */
                    total_count = search_mixed_fstring_for_value(
                        data->items,
                        search_value string->string_value,
                        search_value string->string_length,

                        field_offset,
                        search_value list,

                        data->number_of_items,

30                    operator,

                    results_bitmap);
                }
                else
                {
                    /* strings without ? or * wildcard characters */
                    total_count = search_fstring_array_for_value(
                        data->items,
                        search_value string->string_value,
                        search_value string->string_length,

                        field_offset,
                        search_value list,

                        data->number_of_items,

35                    operator,

                    results_bitmap);
                }
            }
        }
    }
}

```



```

    }
    else
    {
        if (((strchr(search_value.string->string_value '?') != NULL) &&
            ((strchr(search_value.string->string_value '*') != NULL)))
            total_count = search_string_array_for_value(
5         data->items,
            search_value.string->string_value,
            search_value.string->string_length,

            field_offset,
            NULL,

            results_bitmap),
            data->number_of_items,
            else
10         total_count = search_mixed_string_for_value(
            data->items,
            search_value.string->string_value,
            search_value.string->string_length,

            field_offset,
            NULL,

            results_bitmap),
            data->number_of_items
        )
15     break;
    case string_type:
        if (search_value_type == field_value)
        {
            total_count = search_string_field_array(

            data->items,
            data1->items,
            operator,
            data->number_of_items,
            field_size,
            results_bitmap)
20         }
    else
    {
        if (search_value_type == list_value)
        /* we are looking at a list of values */
        {
            /* see if any mixed strings or not */
            if (check_list_for_mixed_strings( search_value.list ))
25         {
                /* mixed strings case */
                total_count = search_mixed_string_for_value(
                data->items,
                search_value.string->string_value,
                search_value.string->string_length,

                operator,
                search_value.list,

                results_bitmap),
                data->number_of_items,
            }
            else
            {
                /* strings without ? or * wildcard characters */
                total_count = search_string_array_for_value(
30                 data->items,
                search_value.string->string_value,
                search_value.string->string_length,

                operator,
                search_value.list,

                results_bitmap);
                data->number_of_items,
            }
        }
    }
35

```

```

    }
    else
    {
        if (((strchr(search_value.string->string_value, '?')) == NULL) &&
            ((strchr(search_value.string->string_value, '*') != NULL))
5         total_count = search_string_array_for_value(
            data->items,
            search_value.string->string_value,
            search_value.string->string_length,
            NULL,
            operator,
            data->number_of_items,
            results_bitmap);
        else
10         total_count = search_mixed_string_for_value(
            data->items,
            search_value.string->string_value,
            search_value.string->string_length,
            NULL,
            operator,
            data->number_of_items,
            results_bitmap);
    }
    break;
15 case br_type:
    if (search_value_type == field_value)
    {
        total_count = search_br_field_array(
            data->items,
            data1->items,
            operator,
            data->number_of_items,
            results_bitmap);
20     }
    else
    {
        if (search_value_type == list_value)
            /* we are looking at a list of values */
        {
            total_count = search_br_array_for_value(
            data->items
25             (unsigned int)0,
             (unsigned int)0,
            search_value.list,
            operator,
            data->number_of_items,
            results_bitmap);
        }
    }
    else
    {
        total_count = search_br_array_for_value(
            data->items,
30             (unsigned int)0,
             (unsigned int)0,
            NULL,
            operator,
            data->number_of_items,
            results_bitmap);
    }
    break;
    case bad_data_type:
        default: break;
35 } /* end of switch */

return(total_count);
}

```

```

/
SEARCH_MF_LARGE_FIELD_ARRAY
/
/      total_count = search_mf_large_field_array(
5 /      array_start,
/      array_start1,
/      operator,
/      input_bitmap,
/      input_bitmap1,
/      results_bitmap).
/
/ Key difference between SEARCH_MF_LARGE_FIELD_ARRAY and the other
/ SEARCH_MF_????_FIELD_ARRAY routine is that array_start and array_start1
/ are declared as integers here and other types in the other routines.
10 /
/ Used, when the values of fields in 2 separate files are compared.
/ ie, SUB01.dollar_variable > SUB02.dollar_variable
/
/ array_start and array_start1 point to large integer arrays of actual
/ data values from each separate file.
/
/ operator is ==, !=, >, >=, <, <=
/
15 / input_bitmap and input_bitmap1 are master_file sized bitmap with the number of
/ bits set that are in each file
/
/ results_bitmap is the bitmap table this routine is developing with bits set
/ when the data item is in array_start and array_start1 and the operator condition
/ is true.
/
/ input_bitmap and input_bitmap1 are of the size of the master_file and
/ define which elements are in each file. To select an item the bit must
/ be set in both input_bitmaps and then the values pointed to by
20 / array_start and array_start1 must match the operator.
/
/ SEARCH_MF_FIELD_ARRAYS( oper ) expands as:
/
/ j = 1.
/ kk = BITMAP_INTEGER_SIZE.
/ for ( i=0 ; i<num_bits ; ++i )
/
/   bit must be set in both input bitmaps and operator must be true
25 /   if ( (temp & j) && (temp1 & j) && (*array_start oper *array_start1) )
/   {
/     *res |= j
/     total_count++
/   }
/   if bit is set in definition bump past data value
/   if ( temp & j )
/     array_start++
/   if ( temp1 & j )
/     array_start1++
30 /   bump to next bit in word and if needed to next word
/   if ( --kk )
/     j <<= 1
/   else
/   {
/     j = 1.
/     kk = BITMAP_INTEGER_SIZE;
/     temp++, temp1++, res++.
/   }
35 / }
/
/ note in above expansion oper
/
/ if ( (temp & j) && (temp1 & j) && (*array_start OPER *array_start1) )
/
/

```

```

5 int search_mf_large_field_array(array_start,array_start1,operator,
                                input_bitmap,input_bitmap1,results_bitmap)
    unsigned int *array_start;
    unsigned int *array_start1;
    enum field_operator operator;
    struct bitmap *input_bitmap;
    struct bitmap *input_bitmap1;
    struct bitmap *results_bitmap;
    {
        unsigned int j;
10    unsigned int total_count = 0;
        unsigned int *temp, *temp1, *res;
        unsigned int i, j, kk, num_bits;

        temp = input_bitmap->start;
        temp1 = input_bitmap1->start;

        res = results_bitmap->start;

        num_bits = results_bitmap->number_of_bits;

15    switch (operator)
    {
        case equal:
            SEARCH_MF_FIELDS_ARRAYS( if (*array_start == *array_start1 && *array_start < MISSING_LARGE_VALUE ));
            break;
        case greater_than:
            SEARCH_MF_FIELDS_ARRAYS( if (*array_start > *array_start1 && *array_start < MISSING_LARGE_VALUE ));
            break;
        case greater_than_equal:
20    SEARCH_MF_FIELDS_ARRAYS( if (*array_start >= *array_start1 && *array_start < MISSING_LARGE_VALUE ));
            break;
        case less_than_equal:
            SEARCH_MF_FIELDS_ARRAYS( if (*array_start <= *array_start1 && *array_start1 < MISSING_LARGE_VALUE ));
            break;
        case less_than:
            SEARCH_MF_FIELDS_ARRAYS( if (*array_start < *array_start1 && *array_start1 < MISSING_LARGE_VALUE ));
            break;
        case not_equal:
25    SEARCH_MF_FIELDS_ARRAYS( if (*array_start != *array_start1 && *array_start < MISSING_LARGE_VALUE &&
                                *array_start1 < MISSING_LARGE_VALUE ));
            break;
        case between:
        default:
            error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_mf_large_field_array");
            break;
    }
    return(total_count);
30 }

/*
** SEARCH_MF_MEDIUM_FIELD_ARRAY
** Identical to SEARCH_MF_LARGE_FIELD_ARRAY above except that
35 ** array_start and array_start1 are declared as unsigned short type
*/

```

```

int search_mf_medium_field_array(array_start,array_start1,operator,
                                input_bitmap,input_bitmap1,results_bitmap)

unsigned short *array_start;
unsigned short *array_start1;
5  enum field_operator operator;
   struct bitmap *input_bitmap;
   struct bitmap *input_bitmap1;
   struct bitmap *results_bitmap;
   {
       unsigned int j;
       unsigned int total_count = 0;
       unsigned int *temp, *temp1, *res;
       unsigned int i, j, kk, num_bits;

10      temp = input_bitmap->start;
       temp1 = input_bitmap1->start;

       res = results_bitmap->start;

       num_bits = results_bitmap->number_of_bits;

       switch (operator)
       {

15      case equal:
           SEARCH_MF_FIELDS_ARRAYS( if (*array_start == *array_start1 && *array_start < MISSING_MEDIUM_VALUE ));
           break;
       case greater_than:
           SEARCH_MF_FIELDS_ARRAYS( if (*array_start > *array_start1 && *array_start < MISSING_MEDIUM_VALUE ));
           break;
       case greater_than_equal:
           SEARCH_MF_FIELDS_ARRAYS( if (*array_start >= *array_start1 && *array_start < MISSING_MEDIUM_VALUE ));
           break;
       case less_than_equal:
20      SEARCH_MF_FIELDS_ARRAYS( if (*array_start <= *array_start1 && *array_start1 < MISSING_MEDIUM_VALUE ));
           break;
       case less_than:
           SEARCH_MF_FIELDS_ARRAYS( if (*array_start < *array_start1 && *array_start1 < MISSING_MEDIUM_VALUE ));
           break;
       case not_equal:
           SEARCH_MF_FIELDS_ARRAYS( if (*array_start != *array_start1 && *array_start < MISSING_MEDIUM_VALUE &&
                                     *array_start1 < MISSING_MEDIUM_VALUE
                                     break;

25      case between:
       default:
           error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_mf_medium_field_array");
           break;

       }
       return(total_count);

```

30

35

280

```

/*
 * SEARCH_MF_SHORT_FIELD_ARRAY
 *
 * Identical to SEARCH_MF_LARGE_FIELD_ARRAY above except that
 * array_start and array_start1 are declared as unsigned short type
 */

int search_mf_short_field_array(array_start,array_start1,operator,
                               input_bitmap,input_bitmap1,results_bitmap)
5  unsigned char *array_start;
  unsigned char *array_start1;
  enum field_operator operator;
  struct bitmap *input_bitmap;
  struct bitmap *input_bitmap1;
  struct bitmap *results_bitmap;
  {
    unsigned int j;
    unsigned int total_count = 0;
    unsigned int *temp, *temp1, *res;
10    unsigned int i, j, kk, num_bits;

    temp = input_bitmap->start;
    temp1 = input_bitmap1->start;

    res = results_bitmap->start;

    num_bits = results_bitmap->number_of_bits;

    switch (operator)
15    {
      case equal:
        SEARCH_MF_FIELDS_ARRAYS( if (*array_start == *array_start1 && *array_start <
          < MISSING_SHORT_VALUE )); break;
      case greater_than:
        SEARCH_MF_FIELDS_ARRAYS( if (*array_start > *array_start1 && *array_start <
          MISSING_SHORT_VALUE )); break;
      case greater_than_equal:
        SEARCH_MF_FIELDS_ARRAYS( if (*array_start >= *array_start1 && *array_start <
          < MISSING_SHORT_VALUE )); break;
      case less_than_equal:
20      SEARCH_MF_FIELDS_ARRAYS( if (*array_start <= *array_start1 && *array_start1 <
          < MISSING_SHORT_VALUE )); break;
      case less_than:
        SEARCH_MF_FIELDS_ARRAYS( if (*array_start < *array_start1 && *array_start1 <
          < MISSING_SHORT_VALUE )); break;
      case not_equal:
        SEARCH_MF_FIELDS_ARRAYS( if (*array_start != *array_start1 && *array_start <
          < MISSING_SHORT_VALUE && *array_start1
          < MISSING_SHORT_VALUE ));
          break;
25      case between:
      default:
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator in search_mf_short_field_array");
          break;
    }
    return(total_count);
  }
30

```

35

```

5  /*
   * SEARCH_MF_BIT_FIELD_ARRAY
   * identical to SEARCH_MF_LARGE_FIELD_ARRAY above except that
   * array_start and array_start1 are declared as unsigned char type
   * AND
   * use == and != cases only and the match statement for bits (which are
   * in unsigned chars) looks bits in array_start and array_start1
10  * only.
   * DJT to SVP - the logic here is shaky, possibly, how are bits stored
   * here, we are using unsigned char pointers and ++ moving to the next one
   * while we successively look at bits 1,2,4,8,... and on up.
   */

/* Note: as of May 93, bits not really supported. changes to this routine and elsewhere likely needed yet */
int search_mf_bit_field_array(array_start, array_start1, operator,
15                          input_bitmap, input_bitmap1, results_bitmap)
    unsigned char *array_start;
    unsigned char *array_start1;
    enum field_operator operator;
    struct bitmap *input_bitmap;
    struct bitmap *input_bitmap1;
    struct bitmap *results_bitmap;
{
    unsigned int j;
    unsigned int total_count = 0;
    unsigned int *temp, *temp1, *res;
20    unsigned int i, jj, kk, num_bits;

    temp = input_bitmap->start;
    temp1 = input_bitmap1->start;

    res = results_bitmap->start;

    num_bits = results_bitmap->number_of_bits;

    switch (operator)
25    {
        case equal:
            BITMAP_LOOP_PART_1( );
            /* bit must be set in both definition bitmaps and in both
            data arrays */
            if ( (*temp & jj) && (*temp1 & jj) &&
                (*array_start & jj) && (*array_start1 & jj) )
                SEARCH_MF_FIELDS_ARRAYS_PART_2( );
            break;

30
35

```

```

/* DJT - SVP - what do we do to pick bits here?. I am choosing
to pick if in both definition maps and both bits in array_start and
in array_start1 are not set - ie. opposite of case equal */
case not_equal:
    BITMAP_LOOP_PART_1(),
    if ( (*temp & jj) && (*temp1 & jj) &&
        !(*array_start & jj) && (*array_start1 & jj) )
        SEARCH_MF_FIELDS_ARRAYS_PART_2();
    break;
case less_than:
case less_than_equal:
case greater_than:
case between:
    error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (between) in search_mf_bit_field_array");
    break;
10  default
    error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (default) in search_mf_bit_field_array");
    break;
    }
    return(total_count);
}

15  /*
    == SEARCH_MF_STRING_FIELD_ARRAY
    == identical to SEARCH_MF_LARGE_FIELD_ARRAY above EXCEPT
    == 1. array_start and array_start1 are declared as unsigned char type
    == 2. length of string field is passed.
    == 3. supports == and != cases only.
    == 4. uses strcmp to decide if a match or not
    */
20  int search_mf_string_field_array(array_start,array_start1,operator,length,
                                input_bitmap,input_bitmap1,results_bitmap)
    unsigned char *array_start;
    unsigned char *array_start1;
    enum field_operator operator;
    int length;
    struct bitmap *input_bitmap;
    struct bitmap *input_bitmap1;
    struct bitmap *results_bitmap;
25  {
    unsigned int j;
    unsigned int total_count = 0;
    unsigned int *temp, *temp1, *res;
    unsigned int i, jj, kk, num_bits;
    unsigned int miss_string_len;

    miss_string_len = length;
    if ( (!strcmp(MISSING_STRING_VALUE)) < length )
30  miss_string_len = 1;

    temp = input_bitmap->start;
    temp1 = input_bitmap1->start;
    res = results_bitmap->start;

    num_bits = results_bitmap->number_of_bits;

35  switch (operator)
    {

```



```

5      case equal:
        BITMAP_LOOP_PART_1();
        if ( (*temp & ij) && (*temp1 & ij) &&
            (strcmp(array_start, array_start1, length) == 0) &&
            (strcmp(array_start, MISSING_STRING_VALUE, miss_string_len) != 0) )
        SEARCH_MF_FIELDS_STRING_ARRAYS_PART_2( );
        break;
        case not_equal:
            /* note: if either array_start or array_start1 string is MISSING_STRING_VALUE we
            dont get a match */
            BITMAP_LOOP_PART_1();
            if ( (*temp & ij) && (*temp1 & ij) &&
                (strcmp(array_start, array_start1, length) != 0) &&
                (strcmp(array_start, MISSING_STRING_VALUE, miss_string_len) != 0) &&
                (strcmp(array_start1, MISSING_STRING_VALUE, miss_string_len) != 0) )
            SEARCH_MF_FIELDS_STRING_ARRAYS_PART_2( );
            break;
            == 0 )

10      case greater_than:
        case less_than_equal:
        case less_than:
        case between:
            error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (between) in search_mf_string_field_array");
            break;
            default:
            error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (default) in search_mf_string_field_array");
            break;

20      }
    return(total_count);

```

```

    case equal:
        BITMAP_LOOP_PART_1();
        if ( (Temp & j) && (Temp1 & j) &&
            (strcmp(array_start, array_start1, length) == 0) &&
            (strcmp(array_start, MISSING_STRING_VALUE, miss_string_len) != 0) )
            SEARCH_MF_FIELDS_STRING_ARRAYS_PART_2();
        break;
5   case not_equal:
        /* note: if either array_start or array_start1 string is MISSING_STRING_VALUE we
        dont get a match */
        BITMAP_LOOP_PART_1();
        if ( (Temp & j) && (Temp1 & j) &&
            (strcmp(array_start, array_start1, length) != 0) &&
            (strcmp(array_start, MISSING_STRING_VALUE, miss_string_len) != 0) &&
            (strcmp(array_start1, MISSING_STRING_VALUE, miss_string_len) != 0) )
            SEARCH_MF_FIELDS_STRING_ARRAYS_PART_2();
        break;
10  case greater_than:
    case less_than_equal:
    case less_than:
    case between:
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (between) in search_mf_string_field_array");
        break;
        default:
            error_handler (INVALID_SWITCH_VALUE, ERROR, operator "operator (default) in search_mf_string_field_array");
15  break;

    }
    return(total_count);
}

/*
-- SEARCH_MF_FSTRING_FIELD_ARRAY
20 -- identical to SEARCH_MF_LARGE_FIELD_ARRAY above EXCEPT
-- 1. array_start and array_start1 are declared as unsigned char type
-- 2. length of string field is passed
-- 3. supports == and != cases only
-- 4. uses strcmp to decide if a match or not
-- 5. f_offset and f_offset1 are passed identifying which fixed_field data tables to use

25 int search_mf_fstring_field_array(array_start, array_start1, operator, length, f_offset,
                                f_offset1, input_bitmap, input_bitmap1, results_bitmap)
    unsigned short *array_start;
    unsigned short *array_start1;
    enum field_operator operator;
    int length;
    int f_offset;
    int f_offset1;
    struct bitmap *input_bitmap;
    struct bitmap *input_bitmap1;
30 struct bitmap *results_bitmap;
{
    unsigned int j;
    unsigned int total_count = 0;
    unsigned int Temp, Temp1, res;
    unsigned int i, j, k, num_bits;

    Temp = input_bitmap->start;
    Temp1 = input_bitmap1->start;
35 res = results_bitmap->start;

    num_bits = results_bitmap->number_of_bits;

```

```

switch (operator)
{
    case equal:
        BITMAP_LOOP_PART_1():
        /* string_n_compare looks at string that is found using array_start.1 as
        indices, using fixed_field table with passed f_offset index */
        if ( (!temp & j) && (!temp1 & j) &&
            (fixed_field[f_offset]->fixed_string[array_start] != 0) &&
            (fixed_field[f_offset1]->fixed_string[array_start1] != 0))
            if (strcmp(fixed_field[f_offset]->fixed_string[array_start]->string,
                fixed_field[f_offset1]->fixed_string[array_start1]->string, length) == 0 )
                SEARCH_MF_FIELDS_ARRAYS_PART_2( ).
            break;

    case not_equal:
        BITMAP_LOOP_PART_1():
        /* string_n_compare looks at string that is found using array_start.1 as
        indices, using fixed_field table with passed f_offset index */
        if ( (!temp & j) && (!temp1 & j) &&
            (fixed_field[f_offset]->fixed_string[array_start] != 0) &&
            (fixed_field[f_offset1]->fixed_string[array_start1] != 0))
            if (strcmp(fixed_field[f_offset]->fixed_string[array_start]->string,
                fixed_field[f_offset1]->fixed_string[array_start1]->string, length) != 0 )
                SEARCH_MF_FIELDS_ARRAYS_PART_2( ).
            break;

    case greater_than:
    case less_than_equal:
    case less_than:
    case between:
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (between) in search_mf_fstring_field_array").
        break;

    default:
        error_handler (INVALID_SWITCH_VALUE, ERROR, operator, "operator (default) in search_mf_fstring_field_array").
        break;
}
return (total_count);
}

/*
-- additional search parameters: map_count is associated with field, and array_start,
-- map_count1, if present, is associated with field1, and array_start1,
-- map_count2, if present, is the purchase to products map count
-- pp_oper_code is the 00-13 purchase_product function code for field, and array_start
-- pp_oper_code1 is the 00-13 purchase_product function code for field1, and array_start1
--
-- Note: for queries that use SUB for one of the variables, that input_bitmap must be the definition
-- bitmap, so that we can correctly step through that array_start data field. As of July 1993, we
-- have the needed bitmap. To use some type of saved SUB query bitmap or a SUB bitmap from a sub query
-- we would need to revise this routine to pass still another bitmap for the SUB definition, (we would
-- use the sub_results bitmap in selecting data items and the SUB01_BITMAP, SUB02... bitmap for the
-- array_start(or 1) data.
--
-- Note: excluded_records counts customers without purchase (or product) records that
-- match no queries, and won't be found in EQUAL or NOT_EQUAL, (or LESS_THAN or
-- GREATER_THAN_EQUAL).
*/

int search_mf_pp_large_field_array(array_start, array_start1, operator,
    input_bitmap, input_bitmap1, results_bitmap,
    map_count, map_count1, map_count2,
    pp_oper_code, pp_oper_code1, sub_flag)

```

```

unsigned int *array_start;
unsigned int *array_start1;
enum field_operator operator;
struct bitmap *input_bitmap;
struct bitmap *input_bitmap1;
struct bitmap *results_bitmap;
unsigned short *map_count;
unsigned short *map_count1;
unsigned short *map_count2;
5 unsigned int pp_oper_code;
  unsigned int pp_oper_code1;
  int sub_flag;
  {
    unsigned int *array_end;
    unsigned int *array_end1;
    MULTI_FILE_PUR_PRD_VARS;

    MULTI_FILE_PUR_PRD_PRELIMS.

10 /* map_count = 0 is cus or sub case (cus or sub to PUR or PRD). One customer record to
    possibly many purchase or product records.

    map_count != 0 is and map_count1 = 0 is PURCHASE or PRODUCTS to CUS or SUB case (the
    many to one case of possibly many purchase or product records compared to each customer record)

    map_count != 0 and map_count1 != 0 is PURCHASE or PRODUCTS to PURCHASE or PRODUCTS (the many
    to many cases that are not currently supported */

15 temp = input_bitmap->start;
    temp1 = input_bitmap1->start;

    res = results_bitmap->start;

    num_bits = results_bitmap->number_of_bits.

    jj = 1;
    kk = BITMAP_INTEGER_SIZE;
    jj1 = 1; /* f1 items are associated with field1 */
    kk1 = BITMAP_INTEGER_SIZE;
20 if ( !map_count ) /* cus or sub case no map counts for initial field */
    {
      for ( i=0; i<num_bits; i++ )
        array_end1 array_start1++ ) mc1++;
      if ( !temp1 && jj1 )
      {
        if ( !mc1 )
        {
          switch ( pp_oper1 )
          {
            case pur_prd_domain
            case any_pur_prd
              for ( . array_start1 < array_end1; array_start1++ )
              {
                cond = 0;
                switch ( operator )
                {
                  F1_F2_INT_JJF1_TEMP1( MISSING_LARGE_VALUE );
                  ERR_INV_SW( search_mf_pp_large_field_array );
                }
                if ( cond )
                {
                  *res |= jj;
                  total_count++;
                  break;
                }
                NEXT_FIELD1_BIT;
              }
            break;
            case first_pur_prd :
              while ( array_start1 < array_end1 )
              {
                if ( jj1 & temp1 ) /* look at 1st defined bit in field1 bitmap */

```

```

cond = 0;
switch ( operator )
{
    F1_F2_INT_DIRECT( MISSING_LARGE_VALUE );
    ERR_INV_SW( search_mf_pp_large_field_array );
}
5      if ( cond )
      {
          *res != jj;
          total_count++;
      }
      break;
}
NEXT_FIELD1_BIT;
array_start1++;
10    }
    break;
case second_pur_prd :    /* required_count is 2 */
case third_pur_prd :    /* required_count is 3 */
    if ( *mc1 >= required_count )
    {
        found = 0;
        while ( array_start1 < array_end1 )
        {
            if ( jff1 & *temp1 )    /* look at bits defined in field1 bitmap */
            {
15                found++;
                if ( found == required_count )
                {
                    cond = 0;
                    switch ( operator )
                    {
                        F1_F2_INT_DIRECT( MISSING_LARGE_VALUE );
                        ERR_INV_SW( search_mf_pp_large_field_array );
                    }
20                    if ( cond )
                    {
                        *res != jj;
                        total_count++;
                    }
                    break;
                }
            }
        }
        NEXT_FIELD1_BIT;
        array_start1++;
25    }
    }
    break;
case two_pur_prd :    /* required_count is 2 */
case multiple_pur_prd :    /* required_count is 2 */
case three_pur_prd :    /* required_count is 3 */
    if ( *mc1 >= required_count )
    {
        for ( found=0; array_start1 < array_end1; array_start1++ )
30        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_INT_JFF1_TEMP1( MISSING_LARGE_VALUE );
                ERR_INV_SW( search_mf_pp_large_field_array );
            }
            if ( cond )
            {
35                found++;
                if ( found >= required_count )
                {
                    *res != jj;
                    total_count++;
                    break;
                }
            }
        }
    }

```

```

        )
        NEXT_FIELD1_BIT.
    )
    break.
case last_pur_prd
    if ( array_start1 < array_end1 )
5      {
        for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start1++;
            NEXT_FIELD1_BIT.
        }
        while ( ind- > 0 )
        {
            if ( jf1 & *temp1 ) /* look at last defined bit */
10          {
                cond = 0.
                switch ( operator )
                {
                    F1_F2_INT_DIRECT( MISSING_LARGE_VALUE ).
                    ERR_INV_SW( search_mf_pp_large_field_array );
                }
                if ( cond )
                {
                    *res |= j;
                    total_count++;
15                }
                break;
            }
        }
        PREV_FIELD1_BIT.
        --array_start1.
    )
    break;
20 case ntl_pur_prd :
    if ( *mc1 >= 2 )
    {
        defined = 0;
        for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start1++;
            NEXT_FIELD1_BIT
        }
        while ( ind- > 0 )
25      {
            if ( jf1 & *temp1 ) /* look at last defined bit */
            {
                defined++;
                if ( defined == 2 ) /* looking at next to last record */
                {
                    cond = 0.
                    switch ( operator )
                    {
                        F1_F2_INT_DIRECT( MISSING_LARGE_VALUE ).
                        ERR_INV_SW( search_mf_pp_large_field_array ).
                    }
                    if ( cond )
                    {
                        *res |= j;
                        total_count++;
30                    }
                    break;
                }
            }
        }
        PREV_FIELD1_BIT.
        --array_start1;
35    }
    )

```

```

    }
    break;
    case last2_pur_prd :      /* required_count is 2 */
    case last5_pur_prd :      /* required_count is 5 */
5   /* marking as found if at least 1 of the last required_count (ie. 2,5) pur_prd records this customer
    meets criteria */
    if ( *mc1 >= required_count )
    {
        defined = 0;
        for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start1++;
            NEXT_FIELD1_BIT;
        }
10    while ( ind-- > 0 )
    {
        if ( jf1 & *temp1 ) /* look at last defined bit */
        {
            defined++;
            cond = 0;
            switch ( operator )
            {
15                F1_F2_INT_DIRECT( MISSING_LARGE_VALUE ),
                ERR_INV_SW( search_mf_pp_large_field_array ),
            }
            if ( cond )
            {
                *res |= j
                total_count++;
                break;
            }
            if ( defined >= required_count ) /* have looked at last 2 (or 5) records */
20            {
                break;
            }
        }
    }
    PREV_FIELD1_BIT;
    --array_start1;
}
break;
25 case avg_pur_prd /* not supporting for field comparisons */
case total_pur_prd
    error_handler (INVALID_SWITCH_VALUE, ERROR, pp_oper1,
        pp_oper1 (avg or total) in search_mf_pp_large_field_array");
    break;
}
else
    excluded_records++;
30 if ( !sub_flag || j & *temp )
    array_start++;

    }
    mc1++;
    for ( ; array_start1 < array_end1; array_start1++ )
    {
        NEXT_FIELD1_BIT;
35    }
    NEXT_FIELD_BIT;
}
else

```

```

/* next logic deals with MANY to ONE cases (PUR or PRD to CUS or SUB, creating PUR or PRD size bitmap */
if ( map_count && !map_count1 ) /* first (array_start) is purchase or product, 2nd (array_start1) is cus or sub */
{
    mc--;
    for ( i=0; i<num_bits; i += *mc ) /* so 1st mc-- will use 1st data item */
    {
        mc++;
        array_end = array_start + *mc;
        if ( *temp1 && !j1 )
        {
            if ( *mc )
            {
                switch ( pp_oper )
                {
                    case pur_prd_domain :
                    case any_pur_prd :
                        for ( ; array_start < array_end; array_start++ )
                        {
                            cond = 0;
                            switch ( operator )
                            {
                                {
                                    F1_F2_INT_JJ_TEMP( MISSING_LARGE_VALUE ),
                                    ERR_INV_SW( search_mf_pp_large_field_array ),
                                }
                                if ( cond )
                                {
                                    *res |= jj;
                                    total_count++;
                                }
                                NEXT_FIELD_BIT;
                            }
                        }
                        break;
                    case first_pur_prd :
                        while ( array_start < array_end )
                        {
                            if ( jj & *temp ) /* look at 1st defined bit in field bitmap */
                            {
                                cond = 0;
                                switch ( operator )
                                {
                                    {
                                        F1_F2_INT_DIRECT( MISSING_LARGE_VALUE ),
                                        ERR_INV_SW( search_mf_pp_large_field_array ),
                                    }
                                    if ( cond )
                                    {
                                        *res |= jj;
                                        total_count++;
                                    }
                                }
                                break;
                            }
                            NEXT_FIELD_BIT;
                            array_start++;
                        }
                    }
                    break;
                    case second_pur_prd : /* required_count is 2 */
                    case third_pur_prd : /* required_count is 3 */
                        if ( *mc >= required_count )
                        {
                            found = 0;
                            while ( array_start < array_end )
                            {
                                if ( jj & *temp ) /* look at bits defined in field bitmap */
                                {
                                    found++;
                                }
                            }
                        }
                }
            }
        }
    }
}

```



```

    if ( found == required_count )
    {
        cond = 0;
        switch ( operator )
        {
            F1_F2_INT_DIRECT( MISSING_LARGE_VALUE );
            ERR_INV_SW( search_mf_pp_large_field_array );
        }
        if ( cond )
        {
            *res |= JJ;
            total_count++;
        }
        break;
    }
    NEXT_FIELD_BIT;
    array_start++;
}
break;
/* different case at purchase/product level than three_pur_prd */
case two_pur_prd :
case multiple_pur_prd :
    if ( *mc >= 2 )
    {
        /* creating purchase level bitmap, mark all that qualify */
        for ( found=0; array_start < array_end; array_start++ )
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_INT_JJ_TEMP( MISSING_LARGE_VALUE );
                ERR_INV_SW( search_mf_pp_large_field_array );
            }
            if ( cond )
            {
                found++;
                if ( found == 1 )
                {
                    sav_res = res;
                    sav_JJ = JJ;
                }
                else
                if ( found == 2 )
                {
                    *sav_res |= sav_JJ;
                    total_count++;
                    *res |= JJ;
                    total_count++;
                }
                else
                if ( found > 2 )
                {
                    *res |= JJ;
                    total_count++;
                }
            }
        }
        NEXT_FIELD_BIT;
    }
}
break;
/* note: at purchase or product level three_pur_prd uses different logic than two_pur_prd case */
case three_pur_prd :
    if ( *mc >= 3 )
    {

```

```

/* creating purchase level bitmap. mark all that qualify */
for ( found=0; array_start < array_end; array_start++ )
{
    cond = 0;
    switch ( operator )
    {
        F1_F2_INT_JJ_TEMP( MISSING_LARGE_VALUE );
        ERR_INV_SW( search_mf_pp_large_field_array );
    }
    if ( cond )
    {
        found++;
        if ( found == 1 )
        {
            sav_res = res;
            sav_jj = jj;
        }
        else
        if ( found == 2 )
        {
            sav_res1 = res;
            sav_jj1 = jj;
        }
        else
        if ( found == 3 )
        {
            *sav_res |= sav_jj;
            *sav_res1 |= sav_jj1;
            *res |= jj;
            total_count += 3;
        }
        else
        if ( found > 3 )
        {
            *res |= jj;
            total_count++;
        }
    }
    NEXT_FIELD_BIT;
}
break;
case last_pur_pro
if ( array_start < array_end )
{
    for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
    {
        array_start++;
        NEXT_FIELD_BIT;
    }
    while ( ind-- > 0 )
    {
        if ( jj & *temp ) /* look at last defined bit */
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_INT_DIRECT( MISSING_LARGE_VALUE );
                ERR_INV_SW( search_mf_pp_large_field_array );
            }
            if ( cond )
            {
                *res |= jj;
                total_count++;
            }
        }
        break;
    }
    PREV_FIELD_BIT;
    --array_start;
}

```

```

    }
    break;
case ntl_pur_prd :
    if ( *mc >= 2 )
    {
        5      defined = 0;
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start++;
            NEXT_FIELD_BIT;
        }
        while ( ind- > 0 )
        {
            10      if ( j & *temp ) /* look at last defined bit */
            {
                defined++;
                if ( defined == 2 ) /* looking at next to last record */
                {
                    cond = 0;
                    switch ( operator )
                    {
                        F1_F2_INT_DIRECT( MISSING_LARGE_VALUE );
                        ERR_INV_SW( search_mf_pp_large_field_array );
                    }
                    15      if ( cond )
                    {
                        *res |= j;
                        total_count++;
                    }
                    break;
                }
            }
            PREV_FIELD_BIT;
            --array_start;
        }
        20      break;
case last2_pur_prd
/* required_count is 2 */
case last5_pur_prd :
/* required_count is 5 */
/* marking as found if at least 1 of the last 2 (or 5) pur_prd records this customer meets criteria */
/* creating purchase level map, mark both of last 2 records if both qualify */
    if ( *mc >= required_count )
    {
        25      defined = 0;
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start++;
            NEXT_FIELD_BIT;
        }
        while ( ind- > 0 )
        {
            30      if ( j & *temp ) /* look at last defined bit */
            {
                defined++;
                cond = 0;
                switch ( operator )
                {
                    F1_F2_INT_DIRECT( MISSING_LARGE_VALUE );
                    ERR_INV_SW( search_mf_pp_large_field_array );
                }
                if ( cond )
                {
                    35      *res |= j;
                    total_count++;
                }
                if ( defined >= required_count ) /* have looked at last 2 (or 5) records */
                {
                    break;
                }
            }
        }
    }

```

```

    }
    PREV_FIELD_BIT.
    --array_start:
    }
    }
    break;
5   case avg_pur_prd:      /* not supporting for field comparisons */
    case total_pur_prd:
        error_handler (INVALID_SWITCH_VALUE, ERROR, pp_oper1
            "pp_oper1 (avg or total) in search_mf_pp_large_field_array");
        break;
    }
    }
    else
        excluded_records++;
10  }
    for ( . array_start < array_end; array_start++ )
    {
        NEXT_FIELD_BIT.
    }
    NEXT_FIELD1_BIT.
    if ( !sub_flag || y1 & temp1 )
        array_start1++;
    )
15  else
    /* as of July 93, we are not doing any many to many multi_file field comparisons cases */
    if ( map_count && map_count1 && !map_count2 ) /* both purchase or product, no purchase to product */
    {
        /* purchase to purchase and product to product would go here. */
    }

20  else
    /* now do above for case of 2 map counts */
    {
        /* purchase to product and product to purchase cases would go here */
    }

    return(total_count);

25  )
    /* identical to search_mf_pp_large_field_array except array_start and array_start1
    array_end, array_end1 are unsigned short data pointer, routine names in error messages
    and uses MISSING_MEDIUM_VALUE */
    int search_mf_pp_medium_field_array(array_start, array_start1, operator,
                                        input_bitmap, input_bitmap1, results_bitmap,
                                        map_count, map_count1, map_count2,
                                        pp_oper_code, pp_oper_code1, sub_flag)

    unsigned short *array_start;
    unsigned short *array_start1;
30  enum field_operator operator;
    struct bitmap *input_bitmap;
    struct bitmap *input_bitmap1;
    struct bitmap *results_bitmap;
    unsigned short *map_count;
    unsigned short *map_count1;
    unsigned short *map_count2;
    unsigned int pp_oper_code;
    unsigned int pp_oper_code1;
35  int sub_flag;
    {
        unsigned short *array_end;
        unsigned short *array_end1;
        MULTI_FILE_PUR_PRD_VARS;

```

```

MULTI_FILE_PUR_PRD_PRELIMS:

temp = input_bitmap->start;
temp1 = input_bitmap1->start;

res = results_bitmap->start;

5 num_bits = results_bitmap->number_of_bits;

j = 1;
kk = BITMAP_INTEGER_SIZE;
jjf1 = 1; /* f1 items are associated with field1 */
kkf1 = BITMAP_INTEGER_SIZE;
if ( lmap_count ) /* cus or sub case, no map counts for 1st field */
{
    for ( i=0; i<num_bits; i++)
    {
10         array_end1 = array_start1 + mc1;
        if ( temp && j )
        {
            if ( mc1 )
            {
                switch ( pp_oper1 )
                {
                    case pur_prd_domain :
                    case any_pur_prd :
15                     for ( ; array_start1 < array_end1; array_start1++)
                     {
                         cond = 0;
                         switch ( operator )
                         {
                             {
                                 F1_F2_INT_JF1_TEMP1( MISSING_MEDIUM_VALUE );
                                 ERR_INV_SW( search_mf_pp_medium_field_array );
                             }
                             if ( cond )
20                             {
                                 *res |= j;
                                 total_count++;
                                 break;
                             }
                             NEXT_FIELD1_BIT;
                         }
                     }
                     break;
                    case first_pur_prd :
                    while ( array_start1 < array_end1 )
25                     {
                         if ( jjf1 & temp1 ) /* look at 1st defined bit in field1 bitmap */
                         {
                             cond = 0;
                             switch ( operator )
                             {
                                 {
                                     F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                                     ERR_INV_SW( search_mf_pp_medium_field_array );
                                 }
                                 if ( cond )
30                                 {
                                     *res |= j;
                                     total_count++;
                                 }
                                 break;
                             }
                             NEXT_FIELD1_BIT;
                             array_start1++;
                         }
                     }
                     break;
                    case second_pur_prd : /* required_count is 2 */
                    case third_pur_prd : /* required_count is 3 */
                    if ( mc1 >= required_count )
                    {
                        found = 0;
                        while ( array_start1 < array_end1 )
                        {
                            if ( jjf1 & temp1 ) /* look at bits defined in field1 bitmap */

```

```

    {
        found++;
        if ( found == required_count )
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                ERR_INV_SW( search_mf_pp_medium_field_array );
            }
            if ( cond )
            {
                *res |= jj;
                total_count++;
            }
            break;
        }
        NEXT_FIELD1_BIT;
        array_start1++;
    }
    break;
case two_pur_prd :      /* required_count is 2 */
case multiple_pur_prd : /* required_count is 2 */
case three_pur_prd :   /* required_count is 3 */
    if ( *mc1 >= required_count )
    {
        for ( found=0; array_start1 < array_end1; array_start1++ )
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_INT_JJF1_TEMP1( MISSING_MEDIUM_VALUE );
                ERR_INV_SW( search_mf_pp_medium_field_array );
            }
            if ( cond )
            {
                found++;
                if ( found >= required_count )
                {
                    *res |= jj;
                    total_count++;
                    break;
                }
            }
        }
        NEXT_FIELD1_BIT;
    }
    break;
case last_pur_prd :
    if ( array_start1 < array_end1 )
    {
        for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start1++;
            NEXT_FIELD1_BIT;
        }
        while ( ind-- > 0 )
        {
            if ( jjf1 & *temp1 ) /* look at last defined bit */
            {
                cond = 0;
                switch ( operator )
                {
                    F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                    ERR_INV_SW( search_mf_pp_medium_field_array );
                }
                if ( cond )
            }

```

```

                                *res != 0
                                total_count++;
                                }
                                break;
                                }
                                PREV_FIELD1_BIT,
                                --array_start1;
5                                }
                                break;
                                case nt1_pur_pro:
                                if ( *mc1 >= 2 )
                                {
                                    defined = 0;
                                    for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
                                    {
                                        array_start1++;
                                        NEXT_FIELD1_BIT;
10                                    }
                                    while ( ind- > 0 )
                                    {
                                        if ( jf1 & *temp1 ) /* look at last defined bit */
                                        {
                                            defined++;
                                            if ( defined == 2 ) /* looking at next to last record */
                                            {
15                                                cond = 0;
                                                switch ( operator )
                                                {
                                                    {
                                                        F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                                                        ERR_INV_SW( search_mf_pp_medium_field_array );
                                                    }
                                                }
                                                if ( cond )
                                                {
20                                                    *res != 0;
                                                    total_count++;
                                                }
                                                break;
                                            }
                                        }
                                        PREV_FIELD1_BIT,
                                        --array_start1;
                                    }
                                }
                                break;
25                                case last2_pur_pro /* required_count is 2 */
                                case last5_pur_pro /* required_count is 5 */
                                /* marking as found if at least 1 of the last 2 (or 5) pur_pro records this customer meets criteria */
                                if ( *mc1 >= required_count )
                                {
                                    defined = 0;
                                    for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
                                    {
                                        array_start1++;
                                        NEXT_FIELD1_BIT;
30                                    }
                                    while ( ind- > 0 )
                                    {
                                        if ( jf1 & *temp1 ) /* look at last defined bit */
                                        {
                                            defined++;
                                            cond = 0;
                                            switch ( operator )
                                            {
35                                                {
                                                    F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                                                    ERR_INV_SW( search_mf_pp_medium_field_array );
                                                }
                                            }
                                        }
                                    }
                                }

```

```

        if ( cond )
        {
            *res |= jj;
            total_count++;
            break;
        }
5      if ( defined >= required_count )      /* have looked at last 2 (or 5) records */
        {
            break;
        }
    }
    PREV_FIELD1_BIT,
    --array_start1;
}
}
break;
10 case avg_pur_prd      /* not supporting for field comparisons */
case total_pur_prd :
    error_handler (INVALID_SWITCH_VALUE, ERROR, pp_oper1,
        "pp_oper1 (avg or total) in search_mf_pp_medium_field_array");
    break;
}
}
else
15   excluded_records++;
   if ( !sub_flag || jj & temp )
       array_start++;

       }
       mc1++;
       for ( . array_start1 < array_end1; array_start1++ )
       {
           NEXT_FIELD1_BIT
       }
20   NEXT_FIELD_BIT,
       }
   }
   else
   if ( map_count && !map_count1 )      /* first (array_start1) is purchase or product. 2nd array_start1 is bus or sub */
   {
       mc--;
       /* so 1st mc++ will use 1st data item */
       for ( i=0; i<num_bits; i++ *mc;
           {
25   mc++;
       array_end = array_start + *mc;
       if ( temp1 && !i1 )
       {
           if ( *mc )
           {
               switch ( pp_oper )
               {
                   case pur_prd_domain :
                   case any_pur_prd :
30   for ( . array_start < array_end; array_start++ )
                   {
                       cond = 0;
                       switch ( operator )
                       {
                           {
                               F1_F2_INT_JJ_TEMP( MISSING_MEDIUM_VALUE ),
                               ERR_INV_SW( search_mf_pp_medium_field_array ),
                           }
                       }
                       if ( cond )
                       {
35   *res |= jj;
                           total_count++;
                       }
                   }
               }
           }
       }
   }
}

```



```

NEXT_FIELD_BIT.
    }
    break;
case first_pur_prd :
5   while ( array_start < array_end )
    {
        if ( jj & temp )           /* look at 1st defined bit in field bitmap */
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                ERR_INV_SW( search_mf_pp_medium_field_array );
            }
10         if ( cond )
            {
                *res != jj;
                total_count++;
            }
            break;
        }
        NEXT_FIELD_BIT;
        array_start++;
15    }
    break;
case second_pur_prd :           /* required_count is 2 */
case third_pur_prd :           /* required_count is 3 */
    if ( *mc >= required_count )
    {
        found = 0;
        while ( array_start < array_end )
        {
20         if ( jj & temp )           /* look at bits defined in field bitmap */
            {
                found++;
                if ( found == required_count )
                {
                    cond = 0;
                    switch ( operator )
                    {
                        F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                        ERR_INV_SW( search_mf_pp_medium_field_array );
                    }
25                 if ( cond )
                    {
                        *res != jj;
                        total_count++;
                    }
                    break;
                }
            }
        }
        NEXT_FIELD_BIT;
        array_start++;
30    }
    break;
/* note at purchase/product level two_pur_prd multiple_pur_prd different logic than
three_pur_prd */
case two_pur_prd :
case multiple_pur_prd :
    if ( *mc >= 2 )
    {
35         /* creating purchase level bitmap. mark all that qualify */
        for ( found=0 ; array_start < array_end; array_start++ )
        {

```

```

cond = 0.
switch ( operator )
{
    F1_F2_INT_JJ_TEMP( MISSING_MEDIUM_VALUE ).
    ERR_INV_SW( search_mf_pp_medium_field_array ).
}
if ( cond )
{
    found++.
    if ( found == 1 )
    {
        sav_res = res.
        sav_jj = jj.
    }
    else
    if ( found == 2 )
    {
        *sav_res |= sav_jj.
        total_count++.
        *res |= jj.
        total_count++.
    }
    else
    if ( found > 2 )
    {
        *res |= jj
        total_count++.
    }
}
}
NEXT_FIELD_BIT.
}
break;
20 case three_pur_prd :           /* here creating purchase/prod map different than two_pur_prd case */
    if ( *mc >= 3 )
    {
        /* creating purchase level bitmap. mark all that qualify */
        for ( found=0, array_start < array_end; array_start++ )
        {
            cond = 0.
            switch ( operator )
            {
                25 F1_F2_INT_JJ_TEMP( MISSING_MEDIUM_VALUE ).
                ERR_INV_SW( search_mf_pp_medium_field_array ).
            }
            if ( cond )
            {
                found++.
                if ( found == 1 )
                {
                    sav_res = res
                    sav_jj = jj.
                }
                30 else
                if ( found == 2 )
                {
                    sav_res1 = res.
                    sav_jj1 = jj.
                }
                else
                35 if ( found == 3 )
                {
                    *sav_res |= sav_jj.
                    *sav_res1 |= sav_jj1.
                    *res |= jj.
                    total_count += 3.
                }
            }
        }
    }
}

```

```

    }
    else
    if ( found > 3 )
    {
        *res |= jj;
        total_count++;
    }
}
NEXT_FIELD_BIT:
}
break;
case last_pur_prd:
    if ( array_start < array_end )
    {
10      for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
      {
          array_start++;
          NEXT_FIELD_BIT;
      }
      while ( ind-- > 0 )
      {
          if ( jj & *temp ) /* look at last defined bit */
          {
15              cond = 0;
              switch ( operator )
              {
                  F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                  ERR_INV_SW( search_mf_pp_medium_field_array );
              }
              if ( cond )
              {
                  *res |= jj;
                  total_count++;
20              }
              break;
          }
          PREV_FIELD_BIT
          --array_start;
      }
    }
    break;
case nll_pur_prd:
    if ( *mc >= 2 )
25    {
        defined = 0;
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start++;
            NEXT_FIELD_BIT;
        }
        while ( ind-- > 0 )
        {
30            if ( jj & *temp ) /* look at last defined bit */
            {
                defined++;
                if ( defined == 2 ) /* looking at next to last record */
                {
                    cond = 0;
                    switch ( operator )
                    {
                        F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                        ERR_INV_SW( search_mf_pp_medium_field_array );
35                    }
                    if ( cond )
                    {
                        *res |= jj;
                        total_count++;
                    }
                }
            }
        }
    }
}

```

```

        break;
    }
    PREV_FIELD_BIT;
    --array_start;
}
break;
5 case last2_pur_prd :      /* required_count is 2 */
  case last5_pur_prd :      /* required_count is 5 */
    /* marking as found if at least 1 of the last 2 (or 5) pur_prd records this customer meets criteria */
    /* creating purchase level map, mark both of last 2 (or 5) records if both qualify */
    if ( *mc >= required_count )
    {
        defined = 0;
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start++;
            10 NEXT_FIELD_BIT;
        }
        while ( ind-- > 0 )
        {
            if ( j & *temp ) /* look at last defined bit */
            {
                defined++;
                cond = 0;
                switch ( operator )
                {
                    15 F1_F2_INT_DIRECT( MISSING_MEDIUM_VALUE );
                      ERR_INV_SW( search_mf_pp_medium_field_array );
                }
                if ( cond )
                {
                    *res |= j;
                    total_count++;
                }
                if ( defined >= required_count ) /* have looked at last 2 (or 5) records */
                {
                    20 break;
                }
            }
        }
        PREV_FIELD_BIT;
        --array_start;
    }
    break;
25 case avg_pur_prd : /* not supporting for field comparisons */
  case total_pur_prd :
    error_handler( INVALID_SWITCH_VALUE, ERROR, pp_oper1,
        "pp_oper1 (avg or total) in search_mf_pp_medium_field_array");
    break;
}
else
    excluded_records++;
30 }
for ( ; array_start < array_end; array_start++ )
{
    NEXT_FIELD_BIT;
}
NEXT_FIELD1_BIT;
if ( !sub_flag || j1 & *temp1 )
    array_start1++;
}
35

```

```

    )
    else
        /* as of July 93, we are not doing any many to many multi_file field comparisons cases */
        if ( map_count && map_count1 && !map_count2 ) /* both purchase or product, no purchase to product */
5      {
        /* purchase to purchase and product to product would go here */
        }
        else
        /* now do above for case of 2 map counts */
        {
        /* purchase to product and product to purchase cases would go here */
        }

10    return(total_count);
}

/* identical to search_mf_pp_large_field_array except array_start and array_start1
array_end, array_end1 are unsigned char data pointer, routine names in error messages
and uses MISSING_SHORT_VALUE */
int search_mf_pp_short_field_array(array_start,array_start1,operator,
15                                     input_bitmap,input_bitmap1,results_bitmap,
                                     map_count,map_count1,map_count2,
                                     pp_oper_code,pp_oper_code1,sub_flag)
    unsigned char *array_start,
    unsigned char *array_start1,
    enum field_operator operator,
    struct bitmap *input_bitmap,
    struct bitmap *input_bitmap1,
    struct bitmap *results_bitmap,
    unsigned short *map_count,
    unsigned short *map_count1,
    unsigned short *map_count2,
20    unsigned int pp_oper_code,
    unsigned int pp_oper_code1,
    int sub_flag;
{
    unsigned char *array_end;
    unsigned char *array_end1;
    MULTI_FILE_PUR_PRD_VARS.
25    MULTI_FILE_PUR_PRD_PRELIMS.

    temp = input_bitmap->start;
    temp1 = input_bitmap1->start;

    res = results_bitmap->start;

    num_bits = results_bitmap->number_of_bits;

    jj = 1;
    kk = BITMAP_INTEGER_SIZE;
30    jjf1 = 1; /* 11 items are associated with field1 */
    kkf1 = BITMAP_INTEGER_SIZE;
    if ( !map_count ) /* cus or sub case, no map counts for 1st field */
    {
        for ( i=0; i<num_bits; i++)
        {
            array_end1 = array_start1 + *mc1;
            if ( !temp && y )
            {
                if ( *mc1 )
                {
                    switch ( pp_oper1 )
                    {

```

```

case pur_prd_domain
case any_pur_prd :
  for ( ; array_start1 < array_end1; array_start1++ )
  {
    cond = 0;
    switch ( operator )
    {
      F1_F2_INT_JJF1_TEMP1( MISSING_SHORT_VALUE );
      ERR_INV_SW( search_mf_pp_short_field_array );
    }
    if ( cond )
    {
      *res |= j;
      total_count++;
      break;
    }
    NEXT_FIELD1_BIT;
  }
  break;
case first_pur_prd :
  while ( array_start1 < array_end1 )
  {
    if ( jjf1 & *temp1 )      /* look at 1st defined bit in field1 bitmap */
    {
      cond = 0;
      switch ( operator )
      {
        F1_F2_INT_DIRECT( MISSING_SHORT_VALUE );
        ERR_INV_SW( search_mf_pp_short_field_array );
      }
      if ( cond )
      {
        *res |= j;
        total_count++;
      }
      break;
    }
    NEXT_FIELD1_BIT;
    array_start1++;
  }
  break;
case second_pur_prd      /* required_count is 2 */
case third_pur_prd      /* required_count is 3 */
  if ( *mc1 >= required_count )
  {
    found = 0;
    while ( array_start1 < array_end1 )
    {
      if ( jjf1 & *temp1 )      /* look at bits defined in field1 bitmap */
      {
        found++;
      }
      if ( found == required_count )
      {
        cond = 0;
        switch ( operator )
        {
          F1_F2_INT_DIRECT( MISSING_SHORT_VALUE );
          ERR_INV_SW( search_mf_pp_short_field_array );
        }
        if ( cond )
        {
          *res |= j;
          total_count++;
        }
        break;
      }
    }
  }

```

305

```

    }
    NEXT_FIELD1_BIT.
    array_start1++;
  }
  break;
case two_pur_prd :      /* required count is 2 */
case multiple_pur_prd : /* required count is 2 */
5 case three_pur_prd :   /* required count is 3 */
  if ( *mc1 >= required_count )
  {
    for ( found=0; array_start1 < array_end1; array_start1++ )
    {
      cond = 0;
      switch ( operator )
      {
        F1_F2_INT_JJF1_TEMP1( MISSING_SHORT_VALUE ).
10      ERR_INV_SW( search_mf_pp_short_field_array );
      }
      if ( cond )
      {
        found++;
        if ( found >= required_count )
        {
          *res |= JJ
          total_count++;
          break;
15        }
      }
    }
    NEXT_FIELD1_BIT
  }
  break;
case last_pur_prd :
  if ( array_start1 < array_end1 )
  {
20    for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
    {
      array_start1++;
      NEXT_FIELD1_BIT.
    }
    while ( ind- > 0 )
    {
      if ( jjf1 & *temp1 ) /* look at last defined bit */
      {
        cond = 0;
        switch ( operator )
        {
          F1_F2_INT_DIRECT( MISSING_SHORT_VALUE ).
          ERR_INV_SW( search_mf_pp_short_field_array );
        }
        if ( cond )
        {
          *res |= JJ
          total_count++;
25        }
        break;
      }
    }
    PREV_FIELD1_BIT
    --array_start1
  }
  break;
case nil_pur_prd
35 if ( *mc1 >= 2 )
  {
    defined = 0;
    for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
    {
      array_start1++;
      NEXT_FIELD1_BIT.
    }
  }

```

```

    }
    while ( ind-- > 0 )
    {
        if ( jf1 & temp1 )      /* look at last defined bit */
        {
            defined++;
            if ( defined == 2 )      /* looking at next to last record */
            {
                cond = 0;
                switch ( operator )
                {
                    F1_F2_INT_DIRECT( MISSING_SHORT_VALUE );
                    ERR_INV_SW( search_mf_pp_short_field_array );
                }
            }

            if ( cond )
            {
                *res |= j;
                total_count++;
            }
            break;
        }
    }
    PREV_FIELD1_BIT;
    --array_start1;
}
break;
case last2_pur_prd :      /* required_count is 2 */
case last5_pur_prd :      /* required_count is 5 */
    /* marking as found if at least 1 of the last 2 (or 5) pur_prd records this customer meets criteria */
    if ( *mc1 >= required_count )
    {
        defined = 0;
        for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start1++;
            NEXT_FIELD1_BIT;
        }
        while ( ind-- > 0 )
        {
            if ( jf1 & temp1 )      /* look at last defined bit */
            {
                defined--;
                cond = 0;
                switch ( operator )
                {
                    F1_F2_INT_DIRECT( MISSING_SHORT_VALUE );
                    ERR_INV_SW( search_mf_pp_short_field_array );
                }
            }
            if ( cond )
            {
                *res |= j;
                total_count++;
                break;
            }
            if ( defined >= required_count )      /* have looked at last 2 (or 5) records */
            {
                break;
            }
        }
    }
    PREV_FIELD1_BIT;
    --array_start1;
}

```



```

    }
    break;
    case avg_pur_prd      /* not supporting for field comparisons */
    case total_pur_prd
        error_handler (INVALID_SWITCH_VALUE, ERROR, pp_oper1
5         "pp_oper1 (avg or total) in search_mf_pp_short_field_array");
        break;
    }
    }
    else
        excluded_records++;
        if ( !sub_flag || j & temp )
            array_start++;
    }
    mc1++;
10   for ( ; array_start1 < array_end1; array_start1++ )
    {
        NEXT_FIELD1_BIT;
    }
    NEXT_FIELD_BIT;
}
else
if ( map_count && !map_count1 )    /* first (array_start) is purchase or product, 2nd array_start1) is cus or sub */
15 {
    mc--;
    for ( i=0; i<num_bits; i += *mc )
    {
        mc++;
        array_end = array_start + *mc;
        if ( temp1 && j1 )
        {
            if ( *mc )
            {
20                 switch ( pp_oper )
                {
                    case pur_prd_domain :
                    case any_pur_prd
                        for ( ; array_start < array_end; array_start++ )
                        {
                            cond = 0;
                            switch ( operator )
                            {
25                                 F1_F2_INT_JJ_TEMP( MISSING_SHORT_VALUE );
                                    ERR_INV_SW( search_mf_pp_short_field_array );
                            }
                            if ( cond )
                            {
                                *res |= j;
                                total_count++;
                            }
                        }
                        NEXT_FIELD_BIT;
                    }
                }
            }
            break;
30   case first_pur_prd :
            while ( array_start < array_end )
            {
                if ( j & temp )
                    /* look at 1st defined bit in field bitmap */
                {
                    cond = 0;
                    switch ( operator )
                    {
35                         F1_F2_INT_DIRECT( MISSING_SHORT_VALUE );
                            ERR_INV_SW( search_mf_pp_short_field_array );
                    }
                    if ( cond )
                    {
                        *res |= j;
                        total_count++;
                    }
                }
            }
        }
    }
}

```

```

        }
        break;
    }
    NEXT_FIELD_BIT;
    array_start++;
}
break;
5 case second_pur_prd :           /* required_count is 2 */
  case third_pur_prd :           /* required_count is 3 */
    if ( *mc >= required_count )
    {
        found = 0;
        while ( array_start < array_end )
        {
            if ( jj & *temp )      /* look at bits defined in field bitmap */
            {
                found++;
                if ( found == required_count )
                {
                    cond = 0;
                    switch ( operator )
                    {
                        F1_F2_INT_DIRECT( MISSING_SHORT_VALUE );
                        ERR_INV_SW( search_mf_pp_shon_field_array );
                    }
                    if ( cond )
                    {
                        *res |= jj;
                        total_count++;
                    }
                    break;
                }
            }
        }
        NEXT_FIELD_BIT;
        array_start++;
    }
    break;
20 }
/* different logic than three_pur_prd case when creating purchase bitmap */
case two_pur_prd :
case multiple_pur_prd :
    if ( *mc >= 2 )
    {
        /* creating purchase level bitmap mark all that qualify */
        for ( found=0, array_start < array_end; array_start++; )
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_INT_JJ_TEMP( MISSING_SHORT_VALUE );
                ERR_INV_SW( search_mf_pp_shon_field_array );
            }
            if ( cond )
            {
                found++;
                if ( found == 1 )
                {
                    sav_res = res;
                    sav_jj = jj;
                }
                else
                if ( found == 2 )
                {
                    *sav_res |= sav_jj;
                    total_count++;
                    *res |= jj;
                    total_count++;
                }
            }
        }
    }
35 }

```

```

        }
        else
        if ( found > 2 )
        {
            *res |= jj;
            total_count++;
        }
    }
    NEXT_FIELD_BIT;
}
break;
/* different logic than two and multiple_pur_prd case when creating purchase bitmap */
case three_pur_prd :
    if ( *mc >= 3 )
    {
        /* creating purchase level bitmap, mark all that qualify */
        for ( found=0; array_start < array_end; array_start++ )
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_INT_JJ_TEMP( MISSING_SHORT_VALUE );
                ERR_INV_SW( search_mf_pp_snon_field_array );
            }
            if ( cond )
            {
                found++;
                if ( found == 1 )
                {
                    sav_res = res;
                    sav_jj = jj;
                }
                else
                if ( found == 2 )
                {
                    sav_res1 = res;
                    sav_jj1 = jj;
                }
                else
                if ( found == 3 )
                {
                    *sav_res |= sav_jj;
                    *sav_res1 |= sav_jj1;
                    *res |= jj;
                    total_count += 3;
                }
                else
                if ( found > 3 )
                {
                    *res |= jj;
                    total_count++;
                }
            }
        }
        NEXT_FIELD_BIT;
    }
}
break;
case last_pur_prd
    if ( array_start < array_end )
    {
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start++;
            NEXT_FIELD_BIT;
        }
    }
}

```

```

    }
    while ( ind-- > 0 )
    {
        if ( j & temp )          /* look at last defined bit */
        {
            cond = 0;
            switch ( operator )
            {
                {
                    F1_F2_INT_DIRECT( MISSING_SHORT_VALUE );
                    ERR_INV_SW( search_mf_pp_short_field_array )
                }
                if ( cond )
                {
                    *res |= j;
                    total_count++;
                }
                break;
            }
        }
        PREV_FIELD_BIT
        --array_start
    }
    break;
case nil_pur_prd
    if ( *mc >= 2 )
    {
        defined = 0;
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start++
            NEXT_FIELD_BIT;
        }
        while ( ind-- > 0 )
        {
            if ( j & temp )          /* look at last defined bit */
            {
                defined++;
                if ( defined == 2 )      /* looking at next to last record */
                {
                    cond = 0;
                    switch ( operator )
                    {
                        {
                            F1_F2_INT_DIRECT( MISSING_SHORT_VALUE );
                            ERR_INV_SW( search_mf_pp_short_field_array );
                        }
                        if ( cond )
                        {
                            *res |= j;
                            total_count++;
                        }
                    }
                    break;
                }
            }
        }
        PREV_FIELD_BIT;
        --array_start;
    }
    break;
case last2_pur_prd :          /* required count is 2 */
case last5_pur_prd :          /* required count is 5 */
    /* marking as found if at least 1 of the last 2 (or 5) pur_prd records this customer meets criteria */
    /* creating purchase level map, mark both of last 2 (or 5) records if both qualify */
    if ( *mc >= required_count )
    {
        defined = 0;
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
    {

```

311

```

    array_start++;
    NEXT_FIELD_BIT.
  }
  while ( ind- > 0 )
  {
    if ( j & temp )      /* look at last defined bit */
    {
      defined++;
      cond = 0
      switch ( operator )
      {
        F1_F2_INT_DIRECT( MISSING_SHORT_VALUE ).
        ERR_INV_SW( search_mt_pp_short_field_array ).
      }
      if ( cond )
      {
        *res |= j.
        total_count++;
      }
      if ( defined >= required_count )      /* have looked at last 2 (or 5) records */
      {
        break
      }
    }
    PREV_FIELD_BIT.
    --array_start.
  }

  break.
  case avg_pur_prd      /* not supporting for field comparisons */
  case total_pur_prd
    error_handler (INVALID_SWITCH_VALUE, ERROR, pp_oper1
    "pp_oper1 (avg or total) in search_mt_pp_short_field_array")
    break.
  )
  else
    excluded_records++.
  }
  for ( ; array_start < array_end; array_start++ )
  {
    NEXT_FIELD_BIT.
  }
  NEXT_FIELD1_BIT
  if ( 'sub_flag || j1 & temp1 )
    array_start1++.
  )

  }
  else
    /* as of July 93, we are not doing any many to many multi_file field comparisons cases */
    if ( map_count && map_count1 && !map_count2 ) /* both purchase or product, no purchase to product */
    {
      /* purchase to purchase and product to product would go here. */
    }
    else
      /* now do above for case of 2 map counts */
      {
        /* purchase to product and product to purchase cases would go here */
      }

  return(total_count).
}

```

```

int search_mf_pp_string_field_array(array_start,array_start1,operator,length,
                                   input_bitmap,input_bitmap1,results_bitmap,
                                   map_count,map_count1,map_count2,
                                   pp_oper_code,pp_oper_code1,sub_flag)
5  unsigned char *array_start;
  unsigned char *array_start1;
  enum field_operator operator;
  int length;
  struct bitmap *input_bitmap;
  struct bitmap *input_bitmap1;
  struct bitmap *results_bitmap;
  unsigned short *map_count;
  unsigned short *map_count1;
  unsigned short *map_count2;
  unsigned int pp_oper_code;
10  unsigned int pp_oper_code1;
  int sub_flag;
  {
    unsigned char *array_end;
    unsigned char *array_end1;
    unsigned int miss_string_len;

    MULTI_FILE_PUR_PRD_VARS;

    MULTI_FILE_PUR_PRD_PRELIMS;
15  miss_string_len = length;
    if ( (strlen(MISSING_STRING_VALUE)) < length )
        miss_string_len = i;

    temp = input_bitmap->start;
    temp1 = input_bitmap1->start;

    res = results_bitmap->start;
20  num_bits = results_bitmap->number_of_bits;

    jj = 1;
    kk = BITMAP_INTEGER_SIZE;
    jjf1 = 1; /* f1 items are associated with field1 */
    kkf1 = BITMAP_INTEGER_SIZE;
    if ( !map_count ) /* cus or sub case, no map counts for 1st field */
    {
        for ( i=0; i<num_bits; i++ )
25  {
            array_end1 = array_start1 + ( *mc1 * length );
            if ( *temp && y )
            {
                if ( *mc1 )
                {
                    switch ( pp_oper1 )
                    {
                        case pur_pro_domain
                        case any_pur_prd
30  for ( , array_start1 < array_end1; array_start1 += length )
                        {
                            cond = 0;
                            switch ( operator )
                            {
                                F1_F2_STRING_JJF1_TEMP1;
                                ERR_INV_STRING_SW( search_mf_pp_string_field_array );
                            }
                        }

                        if ( cond )
35  {
                            *res |= y;
                            total_count++;
                            break;
                        }
                    }
                }
            }
        }
    }

```

```

        NEXT_FIELD1_BIT;
    }
    break;
case first_pur_pro
    while ( array_start1 < array_end1 )
    {
5      if ( jf1 & Temp1)      /* look at 1st defined bit in field1 bitmap */
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_STRING_DIRECT,
                ERR_INV_STRING_SW( search_mf_pp_string_field_array ).
            }
            if ( cond )
10          {
                *res |= jj;
                total_count++;
            }
            break;
        }
        NEXT_FIELD1_BIT;
        array_start1 += length;
    }
    break;
case second_pur_pro :      /* required_count is 2 */
case third_pur_pro :      /* required_count is 3 */
    if ( *mc1 >= required_count )
    {
        found = 0;
        while ( array_start1 < array_end1 )
        {
            if ( jf1 & Temp1) /* look at bits defined in field1 bitmap */
            {
20              found++;
              if ( found == required_count )
              {
                  cond = 0;
                  switch ( operator )
                  {
                      F1_F2_STRING_DIRECT,
                      ERR_INV_STRING_SW
                      ( search_mf_pp_string_field_array )
                  }
25              if ( cond )
              {
                  *res |= jj;
                  total_count++;
              }
              break;
            }
        }
        NEXT_FIELD1_BIT;
        array_start1 += length;
30    }
    }
    break;
case two_pur_pro          /* required count is 2 */
case multiple_pur_pro :   /* required count is 2 */
case three_pur_pro        /* required count is 3 */
    if ( *mc1 >= required_count )
    {
        for ( found=0; array_start1 < array_end1; array_start1 += length )
35      {
            cond = 0;
            switch ( operator )
            {

```

```

F1_F2_STRING_JJF1_TEMP1
ERR_INV_STRING_SW( search_mf_pp_string_field_array )

    if ( cond )
    {
5      found++;
      if ( found >= required_count )
      {
        *res |= JJ;
        total_count++;
        break;
      }
    }
    NEXT_FIELD1_BIT;
10  }
    break;
case last_pur_prd :
  if ( array_start1 < array_end1 )
  {
    for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
    {
      array_start1 += length;
      NEXT_FIELD1_BIT;
15    }
    while ( ind- > 0 )
    {
      if ( jjf1 & *temp1 ) /* look at last defined bit */
      {
        cond = 0;
        switch ( operator )
        {
          F1_F2_STRING_DIRECT
          ERR_INV_STRING_SW( search_mf_pp_string_field_array );
20          if ( cond )
          {
            *res |= JJ;
            total_count++;
          }
          break;
        }
        PREV_FIELD1_BIT;
25      }
      }
    }
    break;
case nll_pur_prd
  if ( *mc1 >= 2 )
  {
    defined = 0;
    for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
    {
      array_start1 += length;
      NEXT_FIELD1_BIT;
30    }
    while ( ind- > 0 )
    {
      if ( jjf1 & *temp1 ) /* look at last defined bit */
      {
        defined++;
        if ( defined == 2 ) /* looking at next to last record */
        {
          cond = 0;
          switch ( operator )
          {
35          }
        }
      }
    }
  }

```


315

```

        {
            F1_F2_STRING_DIRECT.
            ERR_INV_STRING_SW( search_mf_pp_string_field_array ).
        }
        if ( cond )
        {
            *res != jj;
            total_count++;
5          }
          break;
        }
        PREV_FIELD1_BIT;
        array_start1 = length;
    }
    break;
10 case last2_pur_prd :      /* required_count is 2 */
    case last5_pur_prd :      /* required_count is 5 */
        /* marking as found if at least 1 of the last 2 (or 5) pur_prd records this customer meets criteria */
        if ( *mc1 >= required_count )
        {
            defined = 0;
            for ( ind=1; ind < *mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
            {
                array_start1 = length;
15             NEXT_FIELD1_BIT;
            }
            while ( ind-- > 0 )
            {
                if ( jff1 & *temp1 ) /* look at last defined bit */
                {
                    defined++;
                    cond = 0;
                    switch ( operator )
20                 {
                    F1_F2_STRING_DIRECT.
                    ERR_INV_STRING_SW( search_mf_pp_string_field_array )
                }
                if ( cond )
                {
                    *res != jj;
                    total_count++;
                    break;
                }
25             if ( defined >= required_count ) /* have looked at last 2 (or 5) records */
            {
                break;
            }
        }
        PREV_FIELD1_BIT
        array_start1 = length;
    }
    break;
30 case avg_pur_prd :      /* not supporting for field comparisons */
    case total_pur_prd :
        error_handler (INVALID_SWITCH_VALUE, ERROR, pp_oper1,
            "pp_oper1 (avg or total) in search_mf_pp_string_field_array").
        break;
    }
    else
        excluded_records++;
35 if ( !sub_flag || j & *temp )
    array_start++
}
mc1++;

```

```

    for ( ; array_start1 < array_end1; array_start1 += length )
    {
        NEXT_FIELD1_BIT;
    }
    NEXT_FIELD_BIT;
}
5 else
{
    if ( map_count && !map_count1 ) /* first (array_start) is purchase or product, 2nd (array_start1) is cus or sub */
    {
        mc--; /* so 1st mc++ will use 1st data item */
        for ( i=0; i<num_bits; i += *mc )
        {
            mc++;
            array_end = array_start + ( *mc * length );
            if ( *temp1 && jf1 )
            {
10                 if ( *mc )
                    {
                        switch ( pp_oper )
                        {
                            case pur_prd_domain :
                            case any_pur_prd :
                                for ( ; array_start < array_end; array_start += length )
                                {
                                    cond = 0;
15                                     switch ( operator )
                                        {
                                            F1_F2_STRING_JJ_TEMP;
                                            ERR_INV_STRING_SW( search_mf_pp_string_field_array );

                                            if ( cond )
                                                {
                                                    {
20                                                         *res |= j;
                                                            total_count++;
                                                    }
                                                    NEXT_FIELD_BIT;
                                                }
                                                break;
                                            case first_pur_prd :
                                                while ( array_start < array_end )
                                                {
                                                    if ( j & *temp ) /* look at 1st defined bit in field bitmap */
                                                    {
25                                                         cond = 0;
                                                            switch ( operator )
                                                            {
                                                                F1_F2_STRING_DIRECT
                                                                ERR_INV_STRING_SW( search_mf_pp_string_field_array );

                                                                if ( cond )
                                                                {
30                                                                     *res |= j;
                                                                        total_count++;
                                                                }
                                                                break;
                                                            }
                                                            NEXT_FIELD_BIT;
                                                            array_start += length;
                                                        }
                                                        break;
                                                    case second_pur_prd : /* required_count is 2 */
                                                    case third_pur_prd /* required_count is 3 */
35                                                         if ( *mc >= required_count )
                                                            {

```

```

found = 0;
while ( array_start < array_end )
{
    if ( j & Temp ) /* look at bits defined in field bitmap */
    {
        found++;
        if ( found == required_count )
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_STRING_DIRECT.
                ERR_INV_STRING_SW( search_mf_pp_string_field_array );
            }
            if ( cond )
            {
                *res |= j;
                total_count++;
            }
            break;
        }
        NEXT_FIELD_BIT;
        array_start += length;
    }
}
break;

/* creating purchase bitmap, two and multiple are different logic than three */
case two_pur_prd :
case multiple_pur_prd :
    if ( *mc >= 2 )
    {
        /* creating purchase level bitmap mark all that qualify */
        for ( found=0, array_start < array_end, array_start += length )
        {
            cond = 0;
            switch ( operator )
            {
                F1_F2_STRING_JJ_TEMP
                ERR_INV_STRING_ ERR_INV_STRING_SW( search_mf_pp_string_field_array );
            }
            if ( cond )
            {
                found++;
                if ( found == 1 )
                {
                    sav_res = res;
                    sav_jj = jj;
                }
                else
                if ( found == 2 )
                {
                    *sav_res |= sav_jj;
                    total_count++;
                    *res |= jj;
                    total_count++;
                }
                else
                if ( found > 2 )
                {
                    *res |= jj;
                    total_count++;
                }
            }
        }
        NEXT_FIELD_BIT;
    }
}

```

```

break;
case three_pur_prd: /* creating purchase bitmap, two and multiple are different logic than three */
  if ( *mc >= 3 )
  {
    /* creating purchase level bitmap, mark all that qualify */
    for ( found=0; array_start < array_end; array_start += length )
    {
      cond = 0;
      switch ( operator )
      {
        F1_F2_STRING_JJ_TEMP:
        ERR_INV_STRING_SW( search_mf_pp_string_field_array );
        }
      if ( cond )
      {
        found++;
        if ( found == 1 )
        {
          sav_res = res;
          sav_jj = jj;
        }
        else
        {
          if ( found == 2 )
          {
            sav_res1 = res;
            sav_jj1 = jj;
          }
          else
          {
            if ( found == 3 )
            {
              *sav_res |= sav_jj;
              *sav_res1 |= sav_jj1;
              *res |= jj;
              total_count += 3;
            }
            else
            {
              if ( found > 3 )
              {
                *res |= jj;
                total_count++;
              }
            }
          }
        }
      }
      NEXT_FIELD_BIT;
    }
    break;
case last_pur_prd:
  if ( array_start < array_end )
  {
    for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
    {
      array_start += length;
      NEXT_FIELD_BIT;
    }
    while ( ind- > 0 )
    {
      if ( jj & temp ) /* look at last defined bit */
      {
        cond = 0;
        switch ( operator )
        {
          F1_F2_STRING_DIRECT:
          ERR_INV_STRING_SW( search_mf_pp_string_field_array );
          }
      }
    }
  }

```

```

        if ( cond )
        {
            *res |= jj;
            total_count++;
        }
5       break;
    }
    PREV_FIELD_BIT;
                                array_start = length;
}
break;
case ntl_pur_prd :
    if ( *mc >= 2 )
10   {
        defined = 0;
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start += length;
            NEXT_FIELD_BIT;
        }
        while ( ind-- > 0 )
        {
15         if ( jj & *temp ) /* look at last defined bit */
            {
                defined++;
                if ( defined == 2 ) /* looking at next to last record */
                {
                    cond = 0;
                    switch ( operator )
                    {
20                         F1_F2_STRING_DIRECT;
                        ERR_INV_STRING_SW
                        ( search_mi_pp_string_field_array );
                    }
                }
            }
        }
        if ( cond )
        {
            *res |= jj;
            total_count++;
        }
25       break;
    }
    PREV_FIELD_BIT;
                                array_start = length;
}
break;
case last2_pur_prd : /* required_count is 2 */
30 case last5_pur_prd : /* required_count is 5 */
    /* marking as found if at least 1 of the last 2 (or 5) pur_prd records this customer meets criteria */
    /* creating purchase level map, mark both of last 2 (or 5) records if both qualify */
    if ( *mc >= required_count )
    {
        defined = 0;
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start += length;
            NEXT_FIELD_BIT;
35         }
        while ( ind-- > 0 )
        {
            if ( jj & *temp ) /* look at last defined bit */
            {
                defined++;
                cond = 0;
                switch ( operator )
                {

```

320

```

F1_F2_STRING_DIRECT:
ERR_INV_STRING_SW( search_mf_pp_string_field_array )
    }

    if ( cond )
    {
        *res |= j;
        total_count++;
    }
    if ( defined >= required_count ) /* have looked at last 2 records */
    {
        break;
    }
}
PREV_FIELD_BIT:
    array_start -= length;
}
break;
case avg_pur_prd /* not supporting for field comparisons */
case total_pur_prd :
    error_handler (INVALID_SWITCH_VALUE, ERROR, pp_oper,
        "pp_oper (avg or total) in search_mf_pp_string_field_array");
break;
}
}
else
    excluded_records++;
}
for ( ; array_start < array_end; array_start += length )
{
    NEXT_FIELD_BIT:
}
NEXT_FIELD1_BIT:
if ( 'sub_flag || jf1 & Temp1 )
    array_start1 += length;
}
}
else
/* as of July 93 we are not doing any more to many multi_file field comparisons cases */
if ( map_count && map_count1 && map_count2 ) /* both purchase or product, no purchase to product */
{
    /* purchase to purchase and product to product would go here */
}
else
/* now do above for case of 2 map counts */
{
    /* purchase to product and product to purchase cases would go here */
}
return(total_count);
}

int search_mf_pp_string_fieldarray(array_start, array_start1, operator, length, f_offset,
    f_offset1, input_bitmap, input_bitmap1, results_bitmap,
    map_count, map_count1, map_count2,
    pp_oper_code, pp_oper_code1, sub_flag)

unsigned short *array_start;
unsigned short *array_start1;
enum field_operator operator;
int length;
int f_offset;
int f_offset1;

```

```

struct bitmap *input_bitmap;
struct bitmap *input_bitmap1;
struct bitmap *results_bitmap;
unsigned short *map_count;
unsigned short *map_count1;
unsigned short *map_count2;
5 unsigned int pp_oper_code;
  unsigned int pp_oper_code1;
  int sub_flag;
  {
    unsigned short *array_end;
    unsigned short *array_end1;
    MULTI_FILE_PUR_PRD_VARS;
    MULTI_FILE_PUR_PRD_PRELIMS;
10 temp = input_bitmap->start;
    temp1 = input_bitmap1->start;

    res = results_bitmap->start;

    num_bits = results_bitmap->number_of_bits;

    jj = 1;
    kk = BITMAP_INTEGER_SIZE;
15 jjf1 = 1; /* f1 items are associated with field1 */
    kkf1 = BITMAP_INTEGER_SIZE;
    if ( !map_count ) /* cus or sub case, no map counts for 1st field */
    {
      for ( i=0; i<num_bits; i++ )
      {
        array_end1 = array_start1 + *mc1;
        if ( *temp && jj )
        {
20           if ( *mc1 )
           {
             switch ( pp_oper1 )
             {
               case pur_prd_domain
               case any_pur_prd
                 for ( ; array_start1 < array_end1; array_start1++ )
                 {
                   cond = 0;
                   switch ( operator )
                   {
                     F1_F2_FSTRING_JJF1_TEMP1
                     ERR_INV_STRING_SW( search_mf_pp_fstring_field_array );
                   }
                   if ( cond )
                   {
                     *res |= jj;
                     total_count++;
                     break;
                   }
                 }
                 NEXT_FIELD1_BIT;
25           }
           break;
         case first_pur_prd
           while ( array_start1 < array_end1 )
           {
             if ( jjf1 & *temp1 ) /* look at 1st defined bit in field1 bitmap */
             {
               cond = 0;
               switch ( operator )
               {
35                 F1_F2_FSTRING_DIRECT;
                 ERR_INV_STRING_SW( search_mf_pp_fstring_field_array );

```

```

    }
    if ( cond )
    {
        *res |= jj
        total_count++;
    }
    break;
5   }
    NEXT_FIELD1_BIT;
    array_start1++;
}
break;
case second_pur_prd :    /* required_count is 2 */
case third_pur_prd :    /* required_count is 3 */
    if ( *mc1 >= required_count )
    {
10      found = 0;
        while ( array_start1 < array_end1 )
        {
            if ( jjf1 & *temp1 )    /* look at bits defined in field1 bitmap */
            {
                found++;
            }

            if ( found == required_count )
            {
15              cond = 0;
                switch ( operator )
                {
                    {
                        F1_F2_FSTRING_DIRECT;
                        ERR_INV_STRING_SW( search_mf_pp_tstring_field_array );
                    }
                    if ( cond )
                    {
20                      *res |= jj;
                        total_count++;
                    }
                    break;
                }
            }
            NEXT_FIELD1_BIT;
            array_start1++;
        }
    }
    break;
25 case two_pur_prd :    /* required_count is 2 */
    case multiple_pur_prd :    /* required_count is 2 */
    case three_pur_prd :    /* required_count is 3 */
        if ( *mc1 >= required_count )
        {
            for ( found=0, array_start1 < array_end1, array_start1++; )
            {
30              cond = 0;
                switch ( operator )
                {
                    {
                        F1_F2_FSTRING_JJF1_TEMP1;
                        ERR_INV_STRING_SW( search_mf_pp_tstring_field_array );
                    }
                    if ( cond )
                    {
                        found++;
                        if ( found >= required_count )
                        {
35                          *res |= jj;
                              total_count++;
                              break;
                        }
                    }
                }
            }
        }
    }

```



```

        NEXT_FIELD1_BIT.
    )
    }
    break;
case last_pur_prd :
    if ( array_start1 < array_end1 )
5      {
        for ( ind=1, ind < *mc1, ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start1++
            NEXT_FIELD1_BIT
        }
        while ( ind- > 0 )
        {
            if ( jf1 & Temp1 ) /* look at last defined bit */
10          {
                cond = 0
                switch ( operator )
                {
                    F1_F2_FSTRING_DIRECT.
                    ERR_INV_STRING_SW( search_mf_pp_fstring_field_array )
                }
                if ( cond )
                {
                    *res |= j
                    total_count++
20          }
                break;
            }
            PREV_FIELD1_BIT.
            --array_start1
        }
    }
    break;
case ntl_pur_prd :
25   if ( *mc1 >= 2 )
    {
        defined = 0;
        for ( ind=1, ind < *mc1, ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start1++
            NEXT_FIELD1_BIT.
        }
        while ( ind- > 0 )
        {
            if ( jf1 & Temp1 ) /* look at last defined bit */
            {
                defined++
                if ( defined == 2 ) /* looking at next to last record */
                {
                    cond = 0
                    switch ( operator )
                    {
                        F1_F2_FSTRING_DIRECT.
                        ERR_INV_STRING_SW( search_mf_pp_fstring_field_array );
30          }
                    if ( cond )
                    {
                        *res |= j
                        total_count++
                    }
                    break;
                }
            }
        }
    }
35

```

```

    }
    PREV_FIELD1_BIT.
    --array_start1:
  }
5    }
    break;
case last2_pur_prd:      /* required_count is 2 */
case last5_pur_prd:      /* required_count is 5 */
  /* marking as found if at least 1 of the last 2 ( or 5) pur_prd records this customer meets criteria */
  if ( !mc1 >= required_count )
  {
    defined = 0;
    for ( ind=1; ind < !mc1; ind++ ) /* ind=1 to get to last pur rec this customer */
    {
10      array_start1++;
      NEXT_FIELD1_BIT.
    }
    while ( ind > 0 )
    {
      if ( jf1 & !temp1 ) /* look at last defined bit */
      {
        defined++;
        cond = 0;
        switch ( operator )
        {
15          F1_F2_FSTRING_DIRECT.
          ERR_INV_STRING_SW( search_mf_pp_fstring_field_array ).
        }
        if ( cond )
        {
          !res |= j
          total_count++
          break;
20        }
        if ( defined >= required_count ) /* have looked at last 2 (or 5) records */
        {
          break;
        }
      }
      PREV_FIELD1_BIT.
      --array_start1.
    }
    break;
25    case avg_pur_prd /* not supporting for field comparisons */
    case total_pur_prd
      error_handler (INVALID_SWITCH_VALUE_ERROR, pp_oper1,
        "pp_oper1 (avg or total) in search_mf_pp_fstring_field_array").
      break;
    }
  }
  else
    excluded_records++;
30  if ( !sub_flag || j & !temp )
    array_start1++;
  }
  mc1++;
  for ( ; array_start1 < array_end1; array_start1++ )
  {
    NEXT_FIELD1_BIT.
  }
  NEXT_FIELD_BIT.
35  }

```

```

    }
    else
    if ( map_count && !map_count1 )    /* first (array_start) is purchase or product, 2nd (array_start1) is cus or sub */
    {
        mc--;
        for ( i=0; i<num_bits; i += *mc )
        {
            5      mc++;
            array_end = array_start + *mc;
            if ( *temp1 && j!=1 )
            {
                if ( *mc )
                {
                    switch ( pp_oper )
                    {
                        case pur_prd_domain :
                        case any_pur_prd :
                            10      for ( ; array_start < array_end; array_start++ )
                            {
                                cond = 0;
                                switch ( operator )
                                {
                                    {
                                        F1_F2_FSTRING_JJ_TEMP:
                                        ERR_INV_STRING_SW( search_mf_pp_fstring_field_array );
                                    }
                                    if ( cond )
                                    {
                                        *res |= j;
                                        total_count++;
                                    }
                                }
                            }
                            15      NEXT_FIELD_BIT;
                        break;
                    case first_pur_prd :
                        while ( array_start < array_end )
                        {
                            if ( j & *temp )    /* look at 1st defined bit in field bitmap */
                            {
                                20      cond = 0;
                                switch ( operator )
                                {
                                    {
                                        F1_F2_FSTRING_DIRECT:
                                        ERR_INV_STRING_SW( search_mf_pp_fstring_field_array );
                                    }
                                    if ( cond )
                                    {
                                        *res |= j;
                                        total_count++;
                                    }
                                }
                                25      break;
                            }
                        }
                        NEXT_FIELD_BIT;
                        array_start++;
                    }
                    break;
                case second_pur_prd :    /* required_count is 2 */
                case third_pur_prd :    /* required_count is 3 */
                    30      if ( *mc >= required_count )
                    {
                        found = 0;
                        while ( array_start < array_end )
                        {
                            if ( j & *temp )    /* look at bits defined in field bitmap */
                            {
                                found++;
                                if ( found == required_count )
                                {
                                    35      cond = 0;
                                    switch ( operator )
                                    {

```

```

F1_F2_FSTRING_DIRECT.
ERR_INV_STRING_SW( search_mf_pp_fstring_field_array ).
    }
    if ( cond )
    {
5         *res != j.
        total_count++;
    }
    break.
}
NEXT_FIELD_BIT.
array_start++;
}
10 break;
/* creating purchase_bitmap two and multiple_pur_prd use different logic than case three_pur_prd */
case two_pur_prd :
case multiple_pur_prd :
    if ( *mc >= 2 )
    {
        /* creating purchase level bitmap, mark all that qualify */
        for ( found=0 ; array_start < array_end; array_start++ )
        {
15             cond = 0.
            switch ( operator )
            {
                F1_F2_FSTRING_JJ_TEMP.
                ERR_INV_STRING_SW( search_mf_pp_fstring_field_array ).
            }
            if ( cond )
            {
20                 found++
                if ( found == 1 )
                {
                    sav_res = res.
                    sav_jj = jj.
                }
                else
                if ( found == 2 )
                {
                    *sav_res != sav_jj
                    total_count++
                    *res != jj
                    total_count++
25                 }
                else
                if ( found > 2 )
                {
                    *res != jj
                    total_count++
                }
            }
        }
        NEXT_FIELD_BIT
30    }
    }
    break.
/* creating purchase_bitmap two and multiple_pur_prd use different logic than case three_pur_prd */
case three_pur_prd :
    if ( *mc >= 3 )
    {
        /* creating purchase level bitmap, mark all that qualify */
        for ( found=0 ; array_start < array_end; array_start++ )
        {
35             cond = 0;
            switch ( operator )
            {

```

```

F1_F2_FSTRING_JJ_TEMP;
ERR_INV_STRING_SW( search_mf_pp_fstring_field_array );
    }
    if ( cond )
    {
        found++;
        if ( found == 1 )
        {
            sav_res = res;
            sav_jj = jj;
        }
        else
        if ( found == 2 )
        {
            sav_res1 = res;
            sav_jj1 = jj;
        }
        else
        if ( found == 3 )
        {
            *sav_res |= sav_jj;
            *sav_res1 |= sav_jj1;
            *res |= jj;
            total_count += 3;
        }
        else
        if ( found > 3 )
        {
            *res |= jj;
            total_count++;
        }
    }
    NEXT_FIELD_BIT;
}
}
break;
case last_pur_prd :
    if ( array_start < array_end )
    {
        for ( ind=1; ind < *mc; ind++ ) /* ind=1 to get to last pur rec this customer */
        {
            array_start++;
            NEXT_FIELD_BIT;
        }
        while ( ind-- > 0 )
        {
            if ( jj & Temp ) /* look at last defined bit */
            {
                cond = 0;
                switch ( operator )
                {
                    {
                        F1_F2_FSTRING_DIRECT;
                        ERR_INV_STRING_SW( search_mf_pp_fstring_field_array )
                    }
                    if ( cond )
                    {
                        *res |= jj;
                        total_count++;
                    }
                    break;
                }
            }
            PREV_FIELD_BIT;
            --array_start;
        }
    }
}
break;
case ntl_pur_prd :
    if ( *mc >= 2 )
    {

```

```

defined = 0;
for ( ind=1; ind < *mc; ind++) /* ind=1 to get to last pur rec this customer */
{
    array_start++;
    NEXT_FIELD_BIT;
5   }
    while ( ind- > 0 )
    {
        if ( j & *temp ) /* look at last defined bit */
        {
            defined++;
            if ( defined == 2 ) /* looking at next to last record */
            {
                cond = 0;
                switch ( operator )
                {
                    F1_F2_FSTRING_DIRECT;
                    ERR_INV_STRING_SW( search_mf_pp_fstring_field_array );
                }
                if ( cond )
                {
                    *res |= j;
                    total_count++;
                }
            }
            break;
        }
        PREV_FIELD_BIT;
        --array_start;
    }
    break;
20  case last2_pur_prd : /* required_count is 2 */
    case last5_pur_prd : /* required_count is 5 */
        /* marking as found if at least 1 of the last 2 (or 5) pur_prd records this customer meets criteria */
        /* creating purchase level map, mark both of last 2 (or 5) records if both qualify */
        if ( *mc >= required_count )
        {
            defined = 0;
            for ( ind=1; ind < *mc; ind++) /* ind=1 to get to last pur rec this customer */
            {
                array_start++;
                NEXT_FIELD_BIT;
25   }
                while ( ind- > 0 )
                {
                    if ( j & *temp ) /* look at last defined bit */
                    {
                        defined++;
                        cond = 0;
                        switch ( operator )
                        {
                            F1_F2_FSTRING_DIRECT;
                            ERR_INV_STRING_SW( search_mf_pp_fstring_field_array );
                        }
                        if ( cond )
                        {
                            *res |= j;
                            total_count++;
                        }
                    }
                    if ( defined >= required_count ) /* have looked at last 2 (or 5) records */
                    {
                        break;
                    }
                }
            }
        }
35

```

```

    }
    PREV_FIELD_BIT.
    --array_start.
  }
  break;
5      case avg_pur_prd      /* not supporting for field comparisons */
      case total_pur_prd
        error_handler (INVALID_SWITCH_VALUE, ERROR, pp_oper1
          "pp_oper1 (avg or total) in search_mf_pp_string_field_array").
        break;
    }
  }
  else
    excluded_records++;
10      }
      for ( ; array_start < array_end; array_start++)
      {
        NEXT_FIELD_BIT;
      }
      NEXT_FIELD1_BIT;
      if ( !sub_flag || j11 & temp1 )
        array_start1++;
    }
15      }
      else
        /* as of July 93, we are not doing any many to many multi_file field comparisons cases */
        if ( map_count && map_count1 && !map_count2 ) /* both purchase or product, no purchase to product */
        {
          /* purchase to purchase and product to product would go here */
        }
        else
          /* now do above for case of 2 map counts */
          {
20          /* purchase to product and product to purchase cases would go here */
          }
      return(total_count);
    }
    /*
    -- SEARCH_MF_DATASET_ARRAYS
    -- Calls the appropriate routines to build the results based on their field
25    -- types
    --
    --
    --
    */
    int search_mf_dataset_arrays( data,
30      data1,
      field_type,
      field_size,
      field_offset,
      field_offset1,
      operator,
      search_value,
      search_value_type,
      input_bitmap,
      input_bitmap1,
35      results_bitmap)

```

```

struct dataset *data;
struct dataset *data1;
enum data_type field_type;
int field_size;
int field_offset;
int field_offset1;
enum field_operator operator;
union search_item search_value;
enum search_value_item search_value_type;
struct bitmap *input_bitmap;
struct bitmap *input_bitmap1;
struct bitmap *results_bitmap;

(
    unsigned int total_count = 0;

/*
10  svp - dan, this routine is complete. what you need to do is copy mike's routine for
    field comparisons and change the parameter layout around so that it conforms to
    what we have. Our data->nems is data1 and the data1->nems is data2.
    The routine that mike has on a sheet of paper is our search_mf_medium_field_array
    call given below. All the parameters match, the only additional parameter is the
    operator, which is necessary, since currently he is only doing an equal.
*/
15  switch (field_type)
    {
        case dollars
        case floating_point
        case large_integer
            total_count = search_mf_large_field_array(
20
                data->nems,
                data1->nems,
                operator,
                input_bitmap,
                input_bitmap1,
                results_bitmap);

            break;
        case year_month
        case year_month_day
        case medium_integer
            total_count = search_mf_medium_field_array(
25
                data->nems,
                data1->nems,
                operator,
                input_bitmap,
                input_bitmap1,
                results_bitmap);

            break;
        case character
        case small_integer
            total_count = search_mf_short_field_array(
30
                data->nems,
                data1->nems,
                operator,
                input_bitmap,
                input_bitmap1,
                results_bitmap);

            break;
        case string_type
            /* note: must pass field_size (string_length) for compare */
            total_count = search_mf_string_field_array(
35
                data->nems,
                data1->nems,
                operator,
                field_size,
                input_bitmap,
                input_bitmap1,
                results_bitmap);

            break;
        case fixed_string:
            total_count = search_mf_fstring_field_array(
                data->nems,
                data1->nems,

```



```

operator.

5
break;
case bit_type:
    total_count = search_mf_bit_field_array(

10
break;
case bad_data_type:
    default: break;
} /* end of switch */

/* if (copy_bitmaps(results_bitmap, results_bitmap1) == NULL)
    error_handler (BITMAP_NOT_COPY, ERROR, NO_STATUS,
15
    "results_bitmap to ??? in <search_mf_dataset_arrays>").
*/

return(total_count);
}

/*
- SEARCH_MF_PUR_PRD_DATASET_ARRAYS
- Calls the appropriate routines to build the results based on their field
20
- types
-
-
-
*/

int search_mf_pp_dataset_arrays; data
25
data1,
field_type,
field_size
field_offset,
field_offset1,
operator,
search_value,
search_value_type,
input_bitmap,
input_bitmap1,
results_bitmap,
30
map_count,
map_count1,
map_count2,
oper_code,
oper_code1,
sub_flag)

struct dataset *data;
struct dataset *data1;
enum data_type field_type,
35
int field_size;
int field_offset;
int field_offset1;

```

```

enum field_operator operator;
union search_item search_value;
enum search_value_item search_value_type;
5 struct bitmap *input_bitmap;
  struct bitmap *input_bitmap1;
  struct bitmap *results_bitmap;
  unsigned short *map_count;
  unsigned short *map_count1;
  unsigned short *map_count2;
  unsigned int oper_code;
  unsigned int oper_code1;
10 int sub_flag;
  {
    unsigned int total_count = 0;
  }

/*
  svp - dan, this routine is complete what you need to do is copy mike's routine for
    field comparisons and change the parameter layout around so that it conforms to
    what we have. Our data->items is data1 and the data1->items is data2
    The routine that mike has on a sheet of paper is our search_mf_medium_field_array
    call given below. All the parameters match, the only additional parameter is the
15 operator, which is necessary, since currently he is only doing an equal
*/

switch (field_type)
{
    case dollars
    case floating_point
    case large_integer
20     total_count = search_mf_pp_large_field_array(
        data->items,
        data1->items,
        operator,
        input_bitmap,
        input_bitmap1,
        results_bitmap,
        map_count,
        map_count1,
        map_count2,
        oper_code,
        oper_code1,
        sub_flag);

    break;
    case year_month
    case year_month_day
    case medium_integer
25     total_count = search_mf_pp_medium_field_array(
        data->items,
        data1->items,
        operator,
        input_bitmap,
        input_bitmap1,
        results_bitmap,
        map_count,
        map_count1,
        map_count2,
        oper_code,
        oper_code1,
        sub_flag);

    break;
    case character
    case small_integer
30
35

```

```
total_count = search_mf_pp_short_field_array(

5
    break;
10 case string_type:
    /* note: must pass field_size (string_length) for compare */
    total_count = search_mf_pp_string_field_array(

15
    break;
20 case fixed_string:
    total_count = search_mf_pp_fstring_fieldarray(
        data->items,
        data1->items,
        operator,

25
        field_size,
        field_offset,
        field_offset1,
        input_bitmap,
        input_bitmap1,
        results_bitmap,
        map_count,
        map_count1,
        map_count2,
        oper_code,
        oper_code1,
        sub_flag);

30 break;
    /* note: not supporting bit_type for this yet need to add mf_pp routine and
    expand passed parameter list */
    case bit_type
        total_count = search_mf_bit_field_array(

35
        data->items,
        data1->items,
        operator,
        input_bitmap,
        input_bitmap1,
        results_bitmap);
```

```

                    break;
                case bad_data_type:
                    default: break;
5      } /* end of switch */

      if (copy_bitmaps(results_bitmap, results_bitmap1) == NULL)
          error_handler (BITMAP_NOT_COPY, ERROR, NO_STATUS,
              "results_bitmap to ??? in <search_mf_dataset_arrays>")
      */

10     return(total_count);
    }

15

20

25

30

35
```

WHAT IS CLAIMED IS:

1. A computer implemented data retrieval system comprising:

(a) a database server;

said database server including data storage for storing rotated standard relational database records in rows, wherein said columns of data fields across each record are stored contiguously;

(b) a terminal electrically interconnected to said database server for sending queries to said database server for storing, retrieving and updating said contiguously stored data fields,

(c) said database server including a programmed processor for processing said queries received from said terminal to determine which contiguously stored data fields will be accessed, said programmed processor including a bitmap processor for processing said data fields accessed by performing the following steps:

(i) generating a bitmap for each said accessed data field by assigning a bit to each contiguously stored data field entry; and

(ii) processing said bitmaps to determine which records satisfy said query.

2. The computer implemented data retrieval system as in claim 1, wherein said programmed processor processes said queries by performing the following steps:

(a) scanning said query for a determination of which of said contiguously stored data fields will be accessed to answer said query;

(b) accessing queried data fields;

(c) processing said queried data fields to determine which field entry within said contiguously stored data fields satisfy said query.

3. A computer implemented data retrieval system as in claim 1, wherein said program processor further includes a interactive processor, said interactive processor storing a set number of the most recent queries executed by a system user and the bitmaps associated with said most recent queries from said queries to eliminate bitmap processing for generation of bitmaps for common queries.
4. A computer implemented data retrieval system as in claim 1, wherein said queried data fields have columnar identifiers assigned to each of said fields, said identifiers being indexed to a look up table.
5. A method of searching a relational database for records which match a query comprising the steps of:
- (a) parsing said query into a plurality of pieces where each piece relates to only one field of the database;
 - (b) comparing each piece of said query sequentially to the relevant field of all database records;
 - (c) storing the result of each comparison into a bitmap with a bit position corresponding to the particular record;
 - (d) combining the bitmaps into a results bitmap according to the original query to form an aggregated result.
6. The method of claim 5 including the additional step of counting the records which match the query, said counting step performed by a means for converting said results bitmap into an index, indices or array containing the number of bits set.
7. The method of claim 5 where said step of combining the bitmaps into a results bitmap uses a means for operating on a plurality of bits in parallel fashion.

8. The method of claim 5 including the additional step of storing a ring of the most recent bitmaps containing the results of each piece of a query where each node of said ring contains a query piece definition and a resulting bitmap.

9. The method of claim 5 including the additional step of storing a ring of the prior bitmaps containing the results of each piece of a query where each node of said ring contains said query piece definition and said resulting bitmap.

10. The method of claim 8 including the additional step of parsing a new query into a plurality of pieces, comparing each piece to said query piece definition of said results bitmaps stored in said storage ring, and skipping the processing of any piece which matches a bitmap stored in said storage ring.

11. A method of searching a relational direct marketing database organized in a columnar format for records which match a query comprising the steps of:

(a) parsing said query into a plurality of pieces where each piece relates to only one field of the database;

(b) comparing each piece of said query sequentially to the specific fields of each database which are stored in a contiguous manner, wherein all other fields within the database are not reviewed;

(c) storing the results of each comparison into a bitmap, wherein said bitmaps are one dimensional arrays representative of each said query specific field of data.

(d) combining the bitmaps into a results bitmap according to the original query to form an aggregated result.

12. A method of searching a relational database comprising the steps of:

- (a) querying said fields of data by means of a client server interface program, wherein said client server interface program is loaded onto a personal computer where said queries are entered and processed into packets, said packets being sent to a database server;
- (b) receiving said packets by means of a program running on a database server, said database server storing a database where all database records are rotated 90 degrees such that data for each field is stored contiguously across all customers;
- (c) retrieving data from said database corresponding to the field contained in said packets, wherein data from other fields is not retrieved;
- (d) comparing the query from said packets to the data retrieved from said database;
- (e) creating bitmaps indicating the results of said comparisons, wherein said bitmaps are one-dimensional arrays;
- (f) combining said bitmaps in proper sequence to create a results bitmap forming an aggregated result.
- (g) counting the bits in said results bitmap to determine the number of records in said database which match the query;
- (h) sending the query results back to the client server interface program.

13. The method of claim 12 where said client server program includes a means for cutting, copying, pasting and inserting cells, rows and columns of data.

14. The method of claim 12 where said client server program includes a means for accumulating pieces of queries into a total query connected by logical operators.

15. The method of claim 12 where said client server program includes a means for parsing said query into a plurality of pieces where each piece relates to only one field of the database.

16. The method of claim 12 where said client server program includes a spreadsheet allowing different queries to be combined into a single complex query.

FIELDS	CUSTOMER 1	CUSTOMER 2	CUSTOMER 3	CUSTOMER 4	CUSTOMER 5	CUSTOMER IM
STATE	WI	MN	IL	NY	MI	
ZIP	53206	55442	60601	10708	48202	
AGE	19	26	23	29	24	
INCOME	28,000	31,000	29,000	35,000	34,000	
CITY	MILWAUKEE	MINNEAPOLIS	CHICAGO	BROOKLYN	DETROIT	
GENDER	M	F	F	M	F	
.						
.						
.						
FIELD 1000						



FIG. 1

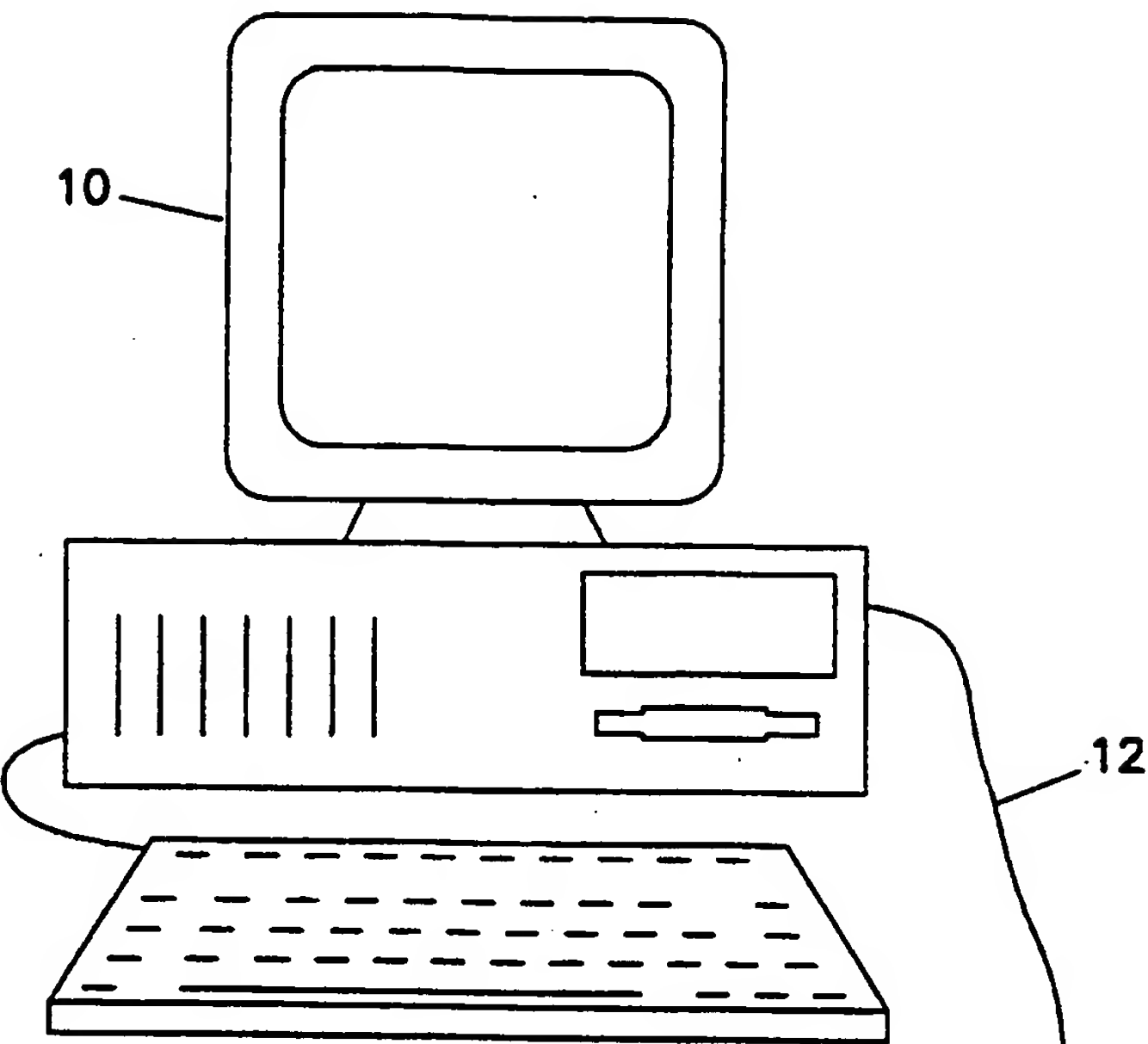
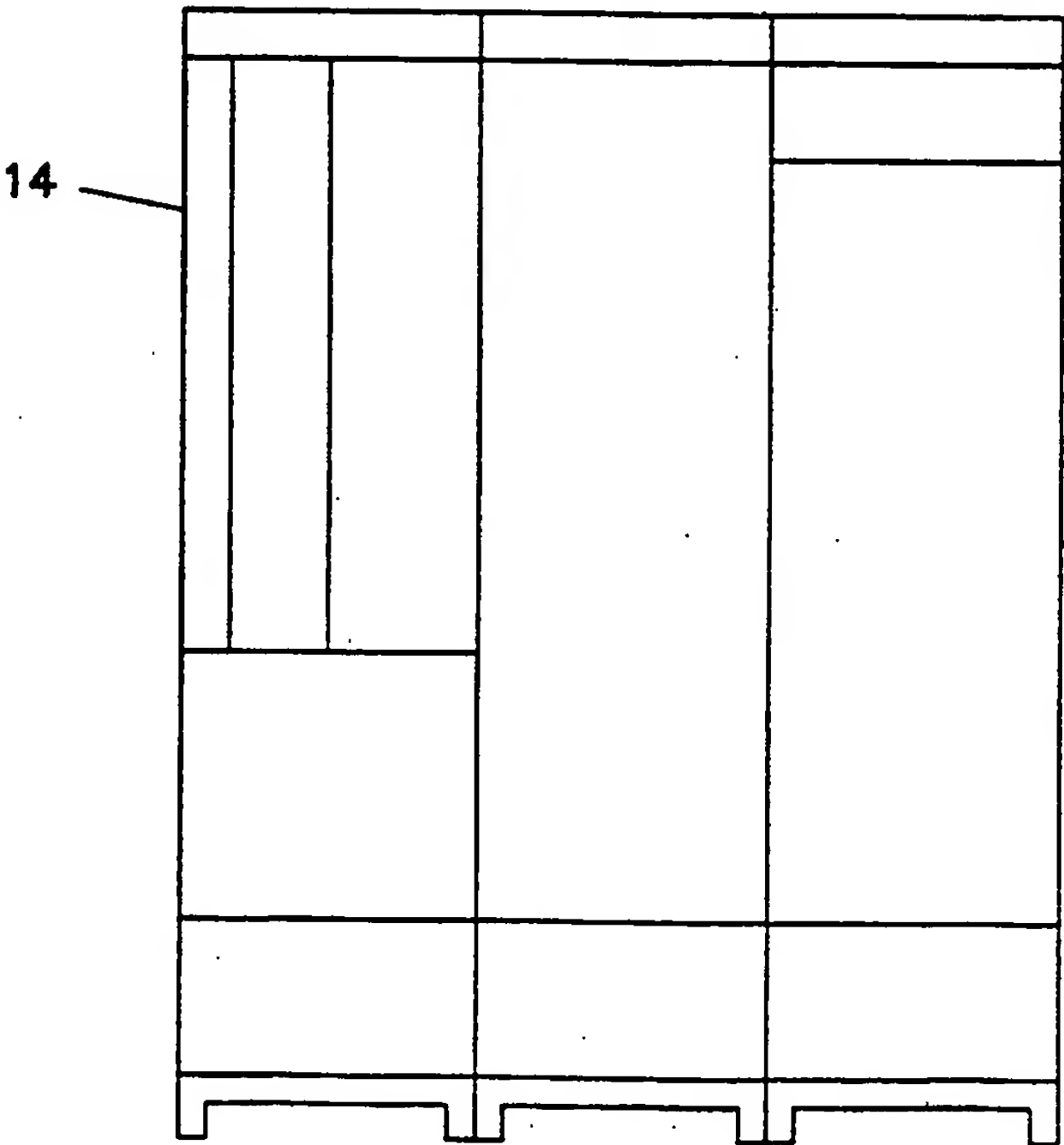
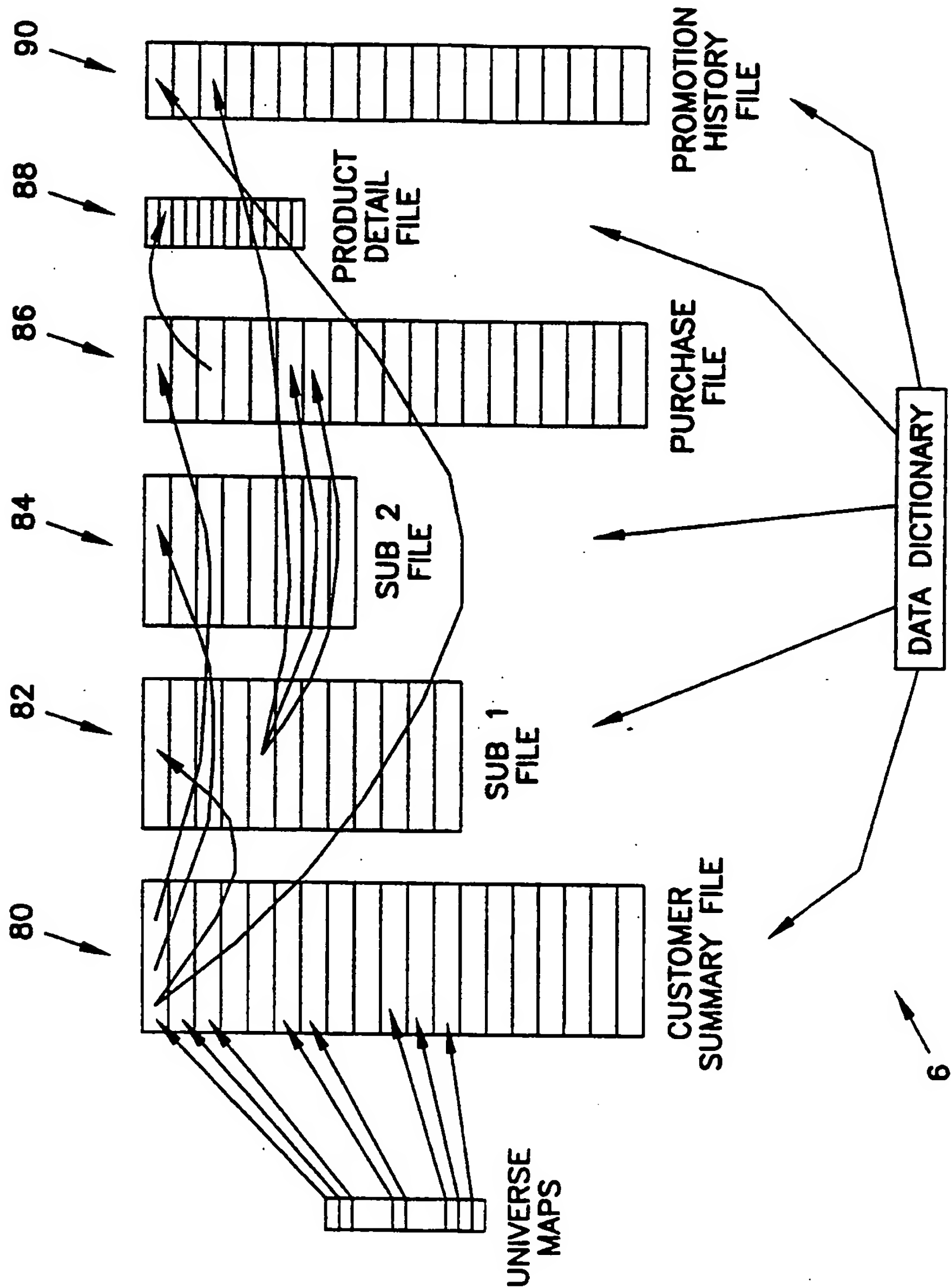


FIG. 2



CUSTOMER	STATE	ZIP	AGE	INCOME	CITY	FIELD 1000
CUSTOMER 1	WI	53206	19	28,000	MILWAUKEE	
CUSTOMER 2	MN	55442	26	31,000	MINNEAPOLIS	
CUSTOMER 3	IL	60601	23	29,000	CHICAGO	
CUSTOMER 4	NY	10708	29	35,000	BROOKLYN	
CUSTOMER 5	MI	48202	24	34,000	DETROIT	
.						
.						
.						
CUSTOMER IM						

FIG. 3



DataBase Link				
File Edit Queries Segmentation Reports Help				
a:\second.job		Active House list	Immediate	
Keycode	Label	A	B	C
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				
26				
27				
28				
29				

8  FIG. 5

Build Query

Edit Queries Help

Query Table

Target Cell

A8

And

Or

End

	Field 1	Operator	Field 2 or Value	Link
1				
2				
3				
4				
5				
6				
↓				

Accept

Quick Query...

Done

Select query table

20  FIG. 6

Distribution

Query Selection

Summary

WIRELESS

CAHILL

SIGNALS

SEASONS

HIGHBRIDGE_RETAIL

↑

↓

↓

↓

Accept

↓

Send...

Fast Fill...

Memorize...

Clear

Done

22

FIG. 7

Two-Way Cross Tab

Query Selection

Summary

WIRELESS

CAHILL

SIGNALS

SEASONS

HIGHBRIDGE_RETAIL

Vertical

Field

ACTIVITY_INDICATOR

Accept

Send...

Fast Fill...

Memorize...

Clear

Done

FIG. 8

24

9/11

Three-Way Cross Tab

Query Selection

Summary

WIRELESS

CAHILL

SIGNALS

SEASONS

HIGHBRIDGE_RETAIL

↓

↓

Accept

↓

Send...

Fast Fill...

Memorize...

Clear

Done

FIG. 9

26

10/11

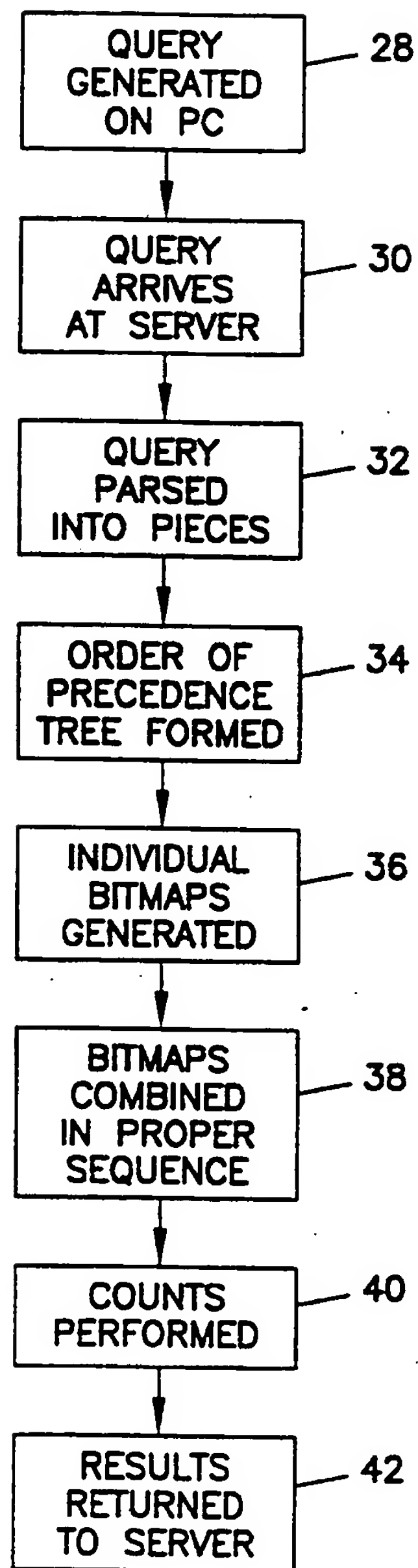


FIG. 10

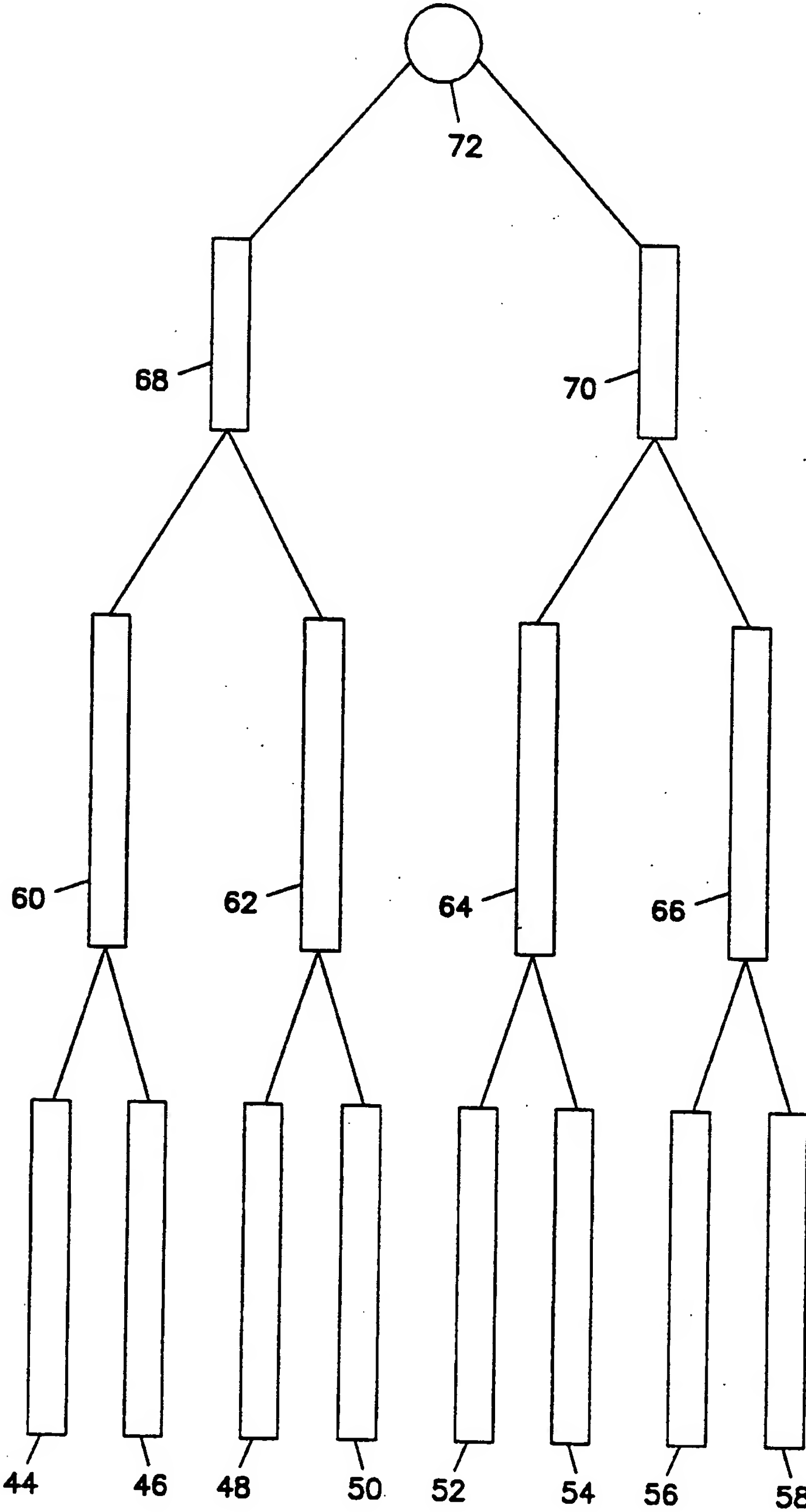


FIG. 11

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/12074

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) : G06F 17/00, 30; 19/00.

US CL : 395/600, 161, 200; 364/227.4, 283.1, 283.4, 284.4

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/600, 161, 200; 364/227.4, 283.1, 283.4, 284.4

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
none

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	DBMS, Volume 4, No. 10, Issued September 1991, J	5-11
----	Winchell, "FoxPro 2.0's Rushmore: here's how FoxPro 2.0's	-----
Y	new technology speeds queries, and when it works best.";	1-4, 12-16
	pages 54-60 especially page 54-57.	
Y	ACM Transactions on Database Systems, Volume 4, No. 4,	1-4, 12-16
	issued December 1979, D. S. Batory, "On Search	
	Transposed Files", pages 531-544, especially page 531-533.	
Y	US, A, 4,751,635 (Kret) 14 June 1988, col 11-19, col 26.	1-4, 12-16
Y	US, A, 3,964,029 (Babb) 15 June 1976, ALL	1-16
A	US, A, 4,785,400 (Kojima et al) 15 November 1988, all	1-16

☒ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:	*T	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
A document defining the general state of the art which is not considered to be part of particular relevance	*X*	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
E earlier document published on or after the international filing date	*Y*	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
L document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	*Z*	document member of the same patent family
O document referring to an oral disclosure, use, exhibition or other means		
P document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 22 DECEMBER 1994	Date of mailing of the international search report 03 APR 1995
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 308-5357	Authorized officer <i>B. Nardew</i> JACK M. CHOULES <i>for</i> Telephone No. (703) 305-9840

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US94/12074

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A, P	US, A, 5,263,159 (Mitsui) 16 November 1993	1-4, 12-16
Y, P	US, A, 5,299,197 (Schlafly) 29 March 1994, col. 3-5.	1-4, 12-16
A	MacWEEK, Volume 6, No. 8, issued 24 February 1992, J. A. Oski, "Muse makes databases sing: Occam Research's easy-to-use data-analysis tool brings flexible reporting to users of corporate databases." pages 37-40.	1-4, 12-16
A	DBMS, Volume 4, No. 10, Issued September 1991, "Rushmore's bald spot"	1-16
A	IEEE, Forth International Conference On Very Large Data Bases, West Berlin, Germany, September 13-15, 1978, R. Ashany, "Application of Sparse Matrix Techniques to Search, Retrieval, Classification and Relationship Analysis in Large Data Base Systems - Sparcom", pages 499-516	1-16

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US94/12074

B. FIELDS SEARCHED

Electronic data bases consulted (Name of data base and where practicable terms used):

APS, DIALOG

search terms: field, column, store, save, contiguous, adjacent, data, transpose, file, record, tables, row, query, invert, bitmap, search, rotate, spreadsheet

THIS PAGE BLANK (USPTO)